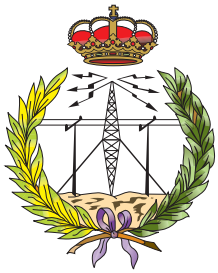

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Virtualización del Sistema Operativo FreeBSD sobre Xen

Guillermo Herrero Delavenay

PROYECTO FIN DE CARRERA

TEMA: Sistemas Operativos, Virtualización

TÍTULO: Virtualización del Sistema Operativo FreeBSD sobre Xen

AUTOR: GUILLERMO HERRERO DELAVENAY

TUTOR: Javier Martín Rueda **VºBº.**

DEPARTAMENTO:

DTE (Departamento de Ingeniería Telemática y Electrónica)

Miembros del tribunal Calificador:

PRESIDENTE*:* Jesús Arriaga García de Andoaín

VOCAL*:* Javier Martín Rueda

VOCAL SECRETARIO*:* Fco. Javier Ramírez Ledesma

Fecha de lectura: 30 de Septiembre de 2014

Calificación:

El Secretario

RESUMEN DEL PROYECTO

Este trabajo es un estudio del funcionamiento del sistema operativo FreeBSD virtualizado sobre el hipervisor de código abierto Xen, ofreciendo una visión completa de la tecnología y realizando un análisis práctico basado en pruebas sintéticas de rendimiento.

El proyecto incluye el diseño y desarrollo de una herramienta de pruebas automatizada y la implementación de un sistema de pruebas dedicado sobre el que se toman medidas de rendimiento para cada uno de los modos de virtualización Xen que soporta FreeBSD actualmente.

Resumen

El sistema operativo FreeBSD soporta distintos modos de virtualización sobre la plataforma Xen. Cada uno usa una técnicas de virtualización distinta, logrando mayor o menor integración con el hipervisor. Actualmente, están soportados en FreeBSD el modo paravirtualizado, virtualizado asistido por hardware y modos híbridos.

Este trabajo consiste fundamentalmente en un estudio práctico de los distintos modos de virtualización Xen soportados en FreeBSD, basándose en pruebas de sintéticas de rendimiento.

Se incluye una comparativa con gráficas de los resultados obtenidos mediante un sistema pruebas automáticas desarrollado en shell script y R.

Abstract

The FreeBSD operative system supports several virtualization modes when used over the Xen platform. Each mode uses a different virtualization technique, achieving different level of integration with the hypervisor. Current supported modes on FreeBSD are paravirtualized mode, hardware virtualization assisted and hybrid modes.

This work is a survey on FreeBSD virtualization over Xen, focused on performance by benchmark testing all supported virtual machine implementations.

The study includes a comparative of the measured test results performed by an automatic testing tool developed on shell and R script.

AGRADECIMIENTOS

A mi padre, por su incansable apoyo y motivación .

A todas las personas que han de alguna forma han hecho este trabajo posible y a aquellas que simplemente se interesaron.

Índice de Contenidos

Índice de Tablas

Índice de Figuras

1. Motivación

FreeBSD ha sido desde hace más de 30 años el sistema operativo de código abierto de referencia para todo tipo de sistemas de producción destacando por su robustez, potencia, sencillez y amplio soporte de dispositivos.

En el momento actual la virtualización ha ganado importancia y sus ventajas son ya imprescindibles. A pesar del excelente soporte hardware que FreeBSD ofrece, actualmente es más práctico utilizarlo sobre hardware virtualizado que directamente sobre una máquina dedicada.

Uno de los entornos de virtualización sobre los que puede ejecutarse FreeBSD es el hipervisor Xen. El proyecto Xen se presentó hace ya una década llevando a práctica el concepto de paravirtualización de una forma efectiva y revolucionaria. Al igual que FreeBSD, Xen es también un proyecto de código abierto y un referente en su ámbito. De entre los hipervisores disponibles actualmente, Xen continua siendo uno de los más avanzados y con más futuro.

Aunque existen otros hipervisores y otros sistemas operativos, la combinación de virtualizar el sistema operativo FreeBSD sobre el hipervisor Xen alberga gran potencial debido a las capacidades y cualidades que poseen los dos sistemas por separado.

FreeBSD, en sus versiones más recientes incluye soporte para funcionar sobre Xen. Sin embargo la integración con el hipervisor no es del todo completa y hasta la fecha acaba de consolidarse. Existen algunas limitaciones y en general hay poca documentación sobre su funcionamiento. Además, la reciente evolución de la tecnología de virtualización hacia modos híbridos ha aumentado el abanico de opciones disponibles a la hora de virtualizar.

Originalmente Xen sólo soportaba la virtualización de sistemas operativos paravirtualizados. Este modo de virtualización, a diferencia de la virtualización total, requiere modificar el núcleo del sistema operativo para retirarle los privilegios de acceso al hardware y hacerlo cooperativo con el hipervisor.

La inclusión de extensiones de virtualización en los procesadores más modernos habilitó la posibilidad de ejecutar sobre Xen sistemas operativos de una forma similar a la paravirtualización pero sin que fuera necesario modificación alguna. Con la posibilidad de ejecutar sistemas no modificados han ido surgiendo modos híbridos que toman funcionalidades de las dos técnicas.

En total existen 5 modos de virtualización de FreeBSD sobre Xen, clasificados en función de la integración que consiguen con el hipervisor:

-Virtualización asistida por hardware.
-Virtualización asistida por hardware con controladores paravirtualizados
-Virtualización asistida con interrupciones y dispositivos paravirtualizados.
-Paravirtualización asistida por hardware.
-Paravirtualización pura.

No se tiene constancia de estudios previos que relacionen el funcionamiento de FreeBSD virtualizado sobre Xen en sus distintas formas soportadas.

Este trabajo tiene como objetivo conocer las diferencias y ventajas que ofrece cada uno de los modos de virtualización sobre Xen soportados actualmente por el sistema operativo FreeBSD, aportando resultados prácticos y comparativas de rendimiento que permitan una valoración objetiva del modo más óptimo en función del uso al que esté destinado.

Identificando los factores más influyentes se pueden diseñar pruebas de rendimiento sintéticas que permitan conocer con precisión el rendimiento del conjunto de máquinas virtuales, para cada tipo de virtualización.

Para obtener la máxima fiabilidad se desarrollará una herramienta de pruebas automatizada capaz de reproducir todos los casos de pruebas en cada una de las máquinas objeto de estudio. La herramienta debe ser capaz de controlar la no interferencia entre pruebas, la extracción de resultados y la generación de gráficas fáciles de interpretar.

2. Virtualización

2.1. Introducción

En computación, el término virtualización se refiere generalmente a la ejecución simultánea de varios sistemas operativos sobre un hardware determinado, compartiendo los recursos de una forma segura y eficiente.

Aunque las técnicas de virtualización fueron creadas inicialmente para solventar problemas básicos de los primeros sistemas de computación modernos, resurgieron años después en el terrero profesional por sus ventajas inherentes de aprovechamiento de recursos y excepcional flexibilidad para gestionarlos. Su primer éxito consistió en la consolidación de servicios que hasta entonces funcionaban en servidores dedicados e infrautilizados, con el consecuente ahorro en costes de operación y energía.

Desde entonces su difusión no ha parado de crecer. Los fabricantes de CPUs ofrecen cada vez mejor soporte y las herramientas software disponibles han proliferado desde unas pocas opciones de pago a un amplio abanico de proyectos de código abierto. Aunque hoy en día es ya imprescindible en el entorno profesional, con un enorme volumen de sistemas a gestionar, la virtualización hace tiempo dejó de ser algo pensado para sistemas de alto rendimiento, para ser directamente, de uso general.

En su etapa más reciente, las mejoras de rendimiento unidas a sus características inherentes de seguridad han propiciado nuevos usos y aplicaciones. El resultado más notable ha sido la aparición de infraestructuras virtualizadas y servicios bajo demanda que extienden el concepto de virtualización a entidades superiores. Estas aplicaciones son la base del llamado “cloud computing”, revolución que abrió la puerta a nuevos modelos de negocio y a los servicios “en la nube” que han transformado la industria de las tecnologías de la información.

Sin embargo, y a pesar de los logros atribuibles a la virtualización, su madurez es muy relativa. Las técnicas y sistemas están en constante evolución y quedan aún multitud de problemas por resolver y mecanismos por mejorar.

Su validez y su utilidad ha quedado demostrada pero, como veremos más adelante, la virtualización actual dista mucho del concepto ideal. Las técnicas de virtualización son o continúan siendo hoy un conjunto de soluciones que solventan limitaciones de base que arrastran las propias arquitecturas de los sistemas que pretenden virtualizar.

2.2. Bases técnicas

Antes de comentar directamente los distintos tipos de virtualización conviene detenerse un momento en los conceptos clave para poder entender mejor qué es una máquina virtual y por qué son necesarias las técnicas de virtualización.

2.2.1. Requerimientos de Popek y Goldberg

Las bases para la virtualización de sistemas fueron formalmente definidas por Popek y Goldberg y expuestas en la ya clásica publicación académica de 1974 “Formal Requirements for Virtualizable Third Generation Architectures” [Popek 1974]. Enunciadas en tiempos y para sistemas más sencillos de los actuales, continúan siendo hoy totalmente válidas a pesar de todo el salto evolutivo.

“A Virtual Machine is the environment created by the virtual machine monitor” [1]

De esta forma tan directa y recurrente, Popek y Goldberg comienzan definiendo el concepto de máquina virtual.

Pero antes, para que una arquitectura sea completamente virtualizable ha de reunir ciertas condiciones. Para ello, clasifican las instrucciones máquina en tres categorías en función de como se comporte la máquina para cada una de ellas:

Instrucciones privilegiadas: Aquellas que sólo serán ejecutadas en modo privilegiado pero que sean capturadas si son ejecutados en otro.

Instrucciones sensibles de control: Aquellas que puedan intenten cambiar la configuración de recursos en el sistema o acceder dispositivos y registros.

Instrucciones sensibles de comportamiento: Aquellas que se comporten de forma distinta dependiendo de la la configuración de recursos.

Al resto de instrucciones se las considera inocuas.

Para que pueda existir un sistema capaz de controlar y monitorizar la ejecución de otros sistemas virtualizados dentro de un mismo entorno, es condición necesaria es que las instrucciones sensibles que estos requieren para funcionar deben ser un subconjunto de las que el sistema privilegiado necesita, manteniendo exclusivas aquellas le permiten retener el control.

El monitor, denominado comúnmente VMM de Virtual Machine Monitor, no es más que un programa de control, totalmente basado en software. Este programa debe reunir las siguientes propiedades:

[1] “Una máquina virtual es el entorno creado por el monitor de máquina virtual”.

Equivalencia. El monitor proporciona un entorno de ejecución que es esencialmente idéntico a de la máquina original.

Rendimiento. La mayoría de las instrucciones que se ejecuten sobre este entorno mostrarán únicamente y como peor caso perdidas leves de velocidad de ejecución.

Control. En todo momento el monitor mantiene el control completo de los recursos del sistema.

Por otro lado, se identifican tres propiedades que deben tener una máquina virtual, la cual estará siempre en ejecución a la vez que el software de control:

Eficiencia. Todas las instrucciones inocuas son ejecutadas directamente por el hardware, sin intervención del monitor.

Control de los recursos. Si un programa intenta acceder directamente a los recursos, se invocará al monitor.

Equivalencia. Todo el control debe hacerse de forma transparente, de forma que el entorno virtualizado funcione de forma exacta e indistinguible a si fuera ejecutado en un entorno propio. Salvando las excepciones de velocidad de ejecución y recursos disponibles, que siempre serán menores.

Cualquier programa de control que satisfaga las tres condiciones puede ser considerado como un VMM. Funcionalmente, cualquier programa en ejecución sobre el entorno que un VMM proporciona, puede denominarse máquina virtual.
El entorno virtualizado está compuesto por la máquina real y el VMM.

2.2.2. Virtualización x86

Desafortunadamente la arquitectura de procesador más popularizada hoy, la arquitectura x86 o IA-32 no fue diseñada teniendo en cuenta la virtualización. El intel 80386 soportaba en su funcionalidad entornos virtuales de procesadores anteriores, pero sin embargo no se ideo la posibilidad de virtualizar su propia arquitectura. En IA32 existen 17 operaciones sensibles que no pueden ser capturadas y por tanto la arquitectura no cumple las condiciones de virtualización de Popek y Goldberg anteriormente descritas. Dado que estas operaciones escriben en registros sin que pueda ser detectado por un monitor de máquina virtual, no es posible mantener el control de una plataforma de virtualización en esta arquitectura.

En realidad esto no significa que sea imposible virtualizar en x86, si no que no lo es directamente o que es más complicado de lo que debería. A pesar de que existen y han existido procesadores diseñados en origen con funciones de virtualización incluso antes que los intel x86, lo cierto es que esta arquitectura por su difusión y coste, siempre ha sido más atractiva frente a otras menos accesibles. Precisamente por su amplia difusión, se ha intentado siempre soportar generaciones anteriores, arrastrando compatibilidades hacia atrás que terminan limitando y añadiendo complejidad a la arquitectura. Aún así la plataforma x86 está tan extendida y tiene tanto mercado que ha habido un enorme esfuerzo para superar sus limitaciones, surgiendo varias técnicas alternativas.

Afortunadamente la situación ahora es mucho mejor que hace unos años. En las nuevas arquitecturas de procesadores de 64 bits se ha puesto especial atención a estas limitaciones y las técnicas necesarias para virtualizar sistemas son mucho más simples. Incluso se ha llegado al punto en que hay funciones de virtualización que se realizan mejor en subsistemas de la CPU que en el propio monitor de virtualización, como son por ejemplo las unidades de gestión de memoria que pueden traducir direcciones virtuales a reales de forma directa y en menor tiempo.

2.3. Técnicas de virtualización

2.3.1. Traducción binaria o virtualización total

Una de las aproximaciones que abordar el problema de la virtualización sobre x86 es la llamada traducción binaria, popularizada inicialmente por VMWare. Su principal ventaja es que permite que la maquina virtual se ejecute en gran parte en modo no privilegiado. Por contra, esta técnica es de las que más afectan a el rendimiento.

La técnica consiste esencialmente en revisar la memoria de la máquina virtual buscando en el flujo de ejecución las instrucciones privilegiadas de forma que estas son luego sustituidas por otras emuladas, dirigiendo de nuevo el control hacia el monitor de virtualización. Las instrucciones siempre se alteran “al vuelo” en vez de procesar y reemplazar todas las operaciones afectadas. Esto es debido a que algunos programas pueden depender o inspeccionar el código internamente desde la propia máquina virtual, como sería el caso de un depurador de código.

La técnica, realmente funciona de una muy forma parecida a como se trabajan los depuradores. Se insertan puntos de parada en el código pero se hace en todas las instrucciones de salto o cambio crítico. Cada vez que una instrucción de este tipo aparece, salta uno de estos puntos de control y se detiene la ejecución. Se pasa al modo privilegiado y el monitor inspecciona las instrucciones de código máquina que van a ejecutarse. De haber una instrucción privilegiada o no segura, ésta es traducida por otra equivalente bajo el control del monitor, de forma totalmente transparente. Por ejemplo el cambio en una instrucción de una dirección de memoria virtual por la adecuada o la redirección al acceder a un dispositivo que la máquina percibe como propio, hacia un fichero o similar.

Como puede apreciarse, la gran ventaja es que el control se pueda hacer siempre desde modo no privilegiado, sin ser necesario salir de este modo hasta que se produce un evento.

Dado que los procesadores x86, a excepción quizá de los más antiguos, ofrecen un conjunto de instrucciones para herramientas de depuración de código, esta técnica de virtualización puede aplicarse prácticamente en cualquier CPU, sin que haya mayor requisito ni funcionalidad relativa a la virtualización.

En cuanto a el rendimiento de este método, éste se más afectado cuando hay muchas operaciones de entrada/salida. Pero en cambio puede ser bastante aceptable cuando se ejecuta un código que contiene un número moderado de instrucciones privilegiadas. El rendimiento típico puede rondar el 80-97% respecto al hardware real. Si bien, estos valores puede ser mucho peores donde haya gran numero de operaciones que necesiten ser traducidas.

2.3.2. Paravirtualización

La paravirtualización aborda el problema x86 de una forma distinta, cambiado el enfoque. Ya que no es posible crear de forma directa entornos de virtuales de la arquitectura, dejar a un lado la complejidad de implementar el entorno virtual ideal y crear el mejor entorno virtual que admita la arquitectura, adaptando luego el sistema operativo a las limitaciones de este pseudo entorno x86.

No se trata por tanto de tener un entorno virtual equivalente sino de crear uno lo suficientemente similar. La paravirtualización, como indica el prefijo *para,* no es virtualización, pero es cercana.

Los entornos paravirtualizados son entonces abstracciones parciales de la capa hardware sobre las que se ejecuta el hipervisor. El sistema operativo debe ser consciente de está en una instancia virtual y comunicarse directamente con el hipervisor de forma similar a como lo haría sobre el hardware. Por lo tanto los sistemas operativos que quieran ejecutarse en entornos virtualizados deben ser adaptados o creados con la paravirtualización en mente.

De alguna forma las limitaciones de la virtualización pasan a ser un problema de diseño de la arquitectura del sistema operativo o del software que va a paravirtualizarse en vez de ser un problema a subsanar completamente por monitor de virtualización, como era caso de la traducción binaria expuesta en el apartado anterior. Con la paravirtualización, las instrucciones x86 no capturables por el VMM en vez de ser tratadas en tiempo de ejecución, deben ser evitadas en el diseño del sistema operativo.

Los cambios en código requeridos para paravirtualizar un sistema operativo existente son bastante abordables. Principalmente el sistema operativo debe pasarse a un modo de ejecución distinto al privilegiado. En la arquitectura x86 existen 4 modos de ejecución de los cuales normalmente sólo se usan los dos más extremos. A estos modos se les conoce como anillos de ejecución siendo el anillo 0 el del modo privilegiado, reservado generalmente por el sistema operativo, y el 3 el asociado a código de usuario.

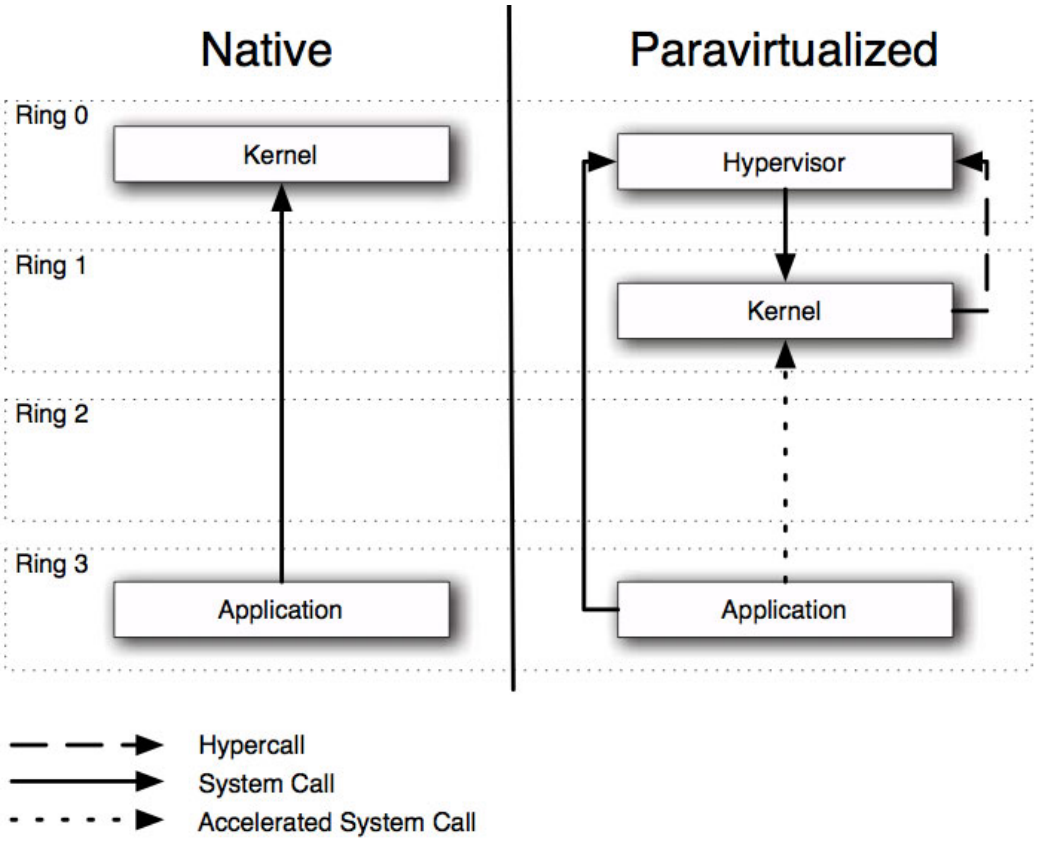

Fig. 2.1 Interacción con el kernel en modo nativo y paravirtualizado.

Cuando el sistema operativo es portado a un entorno paravirtualizado como se hace en Xen, el núcleo del sistema se pasa del 0 al 1. Los programas de usuario o las aplicaciones se mantienen en el nivel de menos privilegios, el 3. El núcleo, desprovisto ahora de todo privilegio debe interactuar con el hipervisor para todas sus operaciones. Mediante llamadas al hipervisor se realizan todas las acciones que en el modo nativo se hacían mediante las llamadas a sistema.

Es importante remarcar que en ningún momento es necesario adaptar las aplicaciones a un sistema paravirtualizado como tampoco lo es para ningún otro sistema virtualizado. Aunque aquí no se mantenga la condición de equivalencia a nivel de arquitectura virtual esto es algo que afecta sólo al sistema operativo. Una vez correctamente adaptado y arrancado, las aplicaciones perciben el mismo entorno de ejecución sin mayores cambios.

Una vez portado, el sistema paravirtualizado resultante puede funcionar de una forma verdaderamente eficiente, con pérdidas típicas de en torno al 1% de rendimiento observadas. Además la paravirtualización ofrece nuevas funcionalidades avanzadas como son la agregación en caliente de CPUs, memoria, nuevos dispositivos e incluso la migración en caliente desde un entorno físico a otro sin parar la ejecución del sistema.

2.3.3. Virtualización asistida por hardware

Históricamente los fabricantes de procesadores han ido incluyendo siempre nuevas y más avanzadas funcionalidades cada vez que presentan la siguiente generación de su arquitectura. En la familia x86 se fue pasando de un procesador de 16 bits con funciones muy limitadas a CPUs con gestión de memoria avanzada, extensiones de cálculo a la vez que se daba el salto a los 32 bit.

Los fabricantes de CPUs compatibles, a veces no implementan las nuevas funciones igual y acaban presentando soluciones distintas como hizo AMD e intel con las extensiones MMX y SEE, que esencialmente hacen algo parecido pero requieren juegos de instrucciones distintas.

En las CPUs más actuales, AMD e Intel han añadido un nuevo conjunto de instrucciones que facilitan de forma considerable la virtualización en la arquitectura x86. Aunque de nuevo han presentado implementaciones distintas, intel VT y AMD-V, la idea que aplican es la misma. Básicamente, tomando el concepto de los anillos o niveles de privilegios, se ha añadido un nivel “-1” de forma que pueda existir un nivel de mayor privilegio que el nivel cero donde hasta ahora residía el núcleo del sistema operativo. De esta forma se puede introducir un hipervisor sin desplazar el sistema operativo del nivel que este espera a la vez que se mantiene control sobre el. Además, las instrucciones sensibles que antes no eran detectables ahora ya pueden ser gestionadas por el hipervisor.

La implementación en si es en realidad algo más complicada pero lo importante es la arquitectura hardware soporta ahora el ser virtualizable de forma totalmente transparente para el sistema operativo, sin que se requiera ningún cambio en este nivel ni sean necesarios procedimientos que suplan las carencias anteriores.

Tanto AMD como Intel han incluido otra serie de funciones relacionadas que facilitan aún más las operaciones involucradas en sistemas virtualizados. Intel ha añade un nuevo modo llamado VMX, invisible al sistema operativo pero disponible para el hipervisor y que permite funciones tales como copiar el estado de completo de la CPU, control avanzado de interrupciones a nivel -1 y 0 y funciones del ciclo de arranque de una máquina virtual.

AMD, aparte de las funciones esenciales para soportar la virtualización hardware incluye gestión de memoria virtual dentro de la propia arquitectura. Mediante esta funcionalidad el hipervisor puede delegar parte del particionado de la memoria virtual en el procesador, evitando el continuo calculo de equivalencias entre las memorias virtuales y física.

Frente al resto de técnicas la virtualización asistida por hardware, también denominada HVM (Hardware Virtualized Machine) tiene sus ventajas y desventajas. Al contrario que la paravirtualización, puede ejecutar sistemas operativos sin ninguna modificación, lo cual puede ser ventajoso para sistemas propietarios o heredados cuyo código no pueda alterarse. Por contra, un sistema así no puede aprovechar ninguna otra optimización disponible por lo que su rendimiento y flexibilidad serían comparativamente reducidos.

2.3.4. Virtualización híbrida

La virtualización híbrida se plantea como una combinación de técnicas de paravirtualización unidas a funciones de virtualización asistida por hardware con el fin de obtener lo mejor de los dos mundos.

La aproximación más simple de un sistema híbrido es el ejemplo de los controladores de dispositivo paravirtualizados que pueden instalarse en sistemas operativos instalados en máquinas HVM. Las máquinas virtualizadas por hardware sufren grandes perdidas de rendimiento en operaciones de entrada/ salida debido a que dependen de dispositivos emulados. En cambio, en los entornos paravirtualizados los controladores de dispositivo son entidades muy sencillas y eficientes, que un rendimiento típico muy alto. Sin que sea necesario modificar el sistema operativo, existe la posibilidad de hacer que una máquina HVM mejore su rendimiento de entrada/salida si se consigue instalar un controlador paravirtualizado. Esta combinación aunque no es el mejor escenario posible, es capaz de mejorar el rendimiento acercándose al rendimiento que el mismo dispositivo puede dar en un entorno paravirtualizado ideal.

Pero el potencial de la virtualización híbrida es la creación de nuevas técnicas de virtualización evolucionadas de mejor rendimiento. Existen al menos dos aproximaciones distintas para crear entornos de virtualización híbrida

Por un lado está lo que se ha llamado paravirtualización asistida por hardware (PVH) y por un lado y por otro, la virtualización asistida por hardware híbrida (PVHVM). [2011 Nakajima].

El concepto de ambos se basa en ejecutar una sistema paravirtualizado dentro de un contenedor HVM para poder utilizar las funciones con soporte hardware asociadas.

La paravirtualización asistida por hardware se beneficia de tener menos latencia en la traducción del mapa de memoria gracias al gestor de memoria virtual hardware y por otro lado obtienen mejoras derivadas de una implementación más efectiva de las llamadas de sistema, al devolver el núcleo paravirtualizado del sistema al nivel 0, de forma que el hipervisor ya no intervenga en todas las llamadas.

En la virtualización asistida por hardware híbrida se obtienen ventajas similares pero se parte de una máquina virtual asistida por hardware a la que se le añaden funciones paravirtualizadas, mejorando la integración con el hipervisor y los rendimientos derivados de una mejor gestión de las interrupciones.

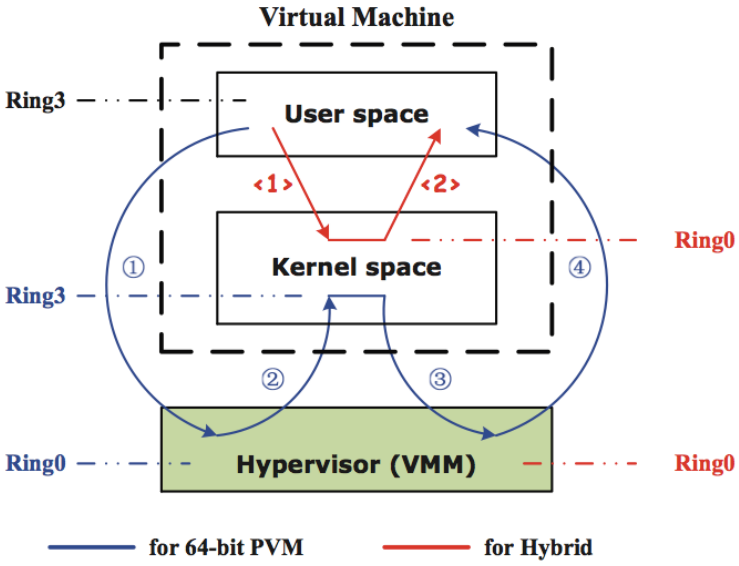

Fig. 2.2 Llamadas a sistema en máquina paravirtualizada (azul) vs híbrida (rojo).

2.3.5. Virtualización a nivel de sistema operativo

Aunque no sea una técnica de virtualización al estilo de las anteriores aquí citadas, intrínsecamente ligadas al hardware y a la problemática de la arquitectura x86, existen otras soluciones alternativas que funcionalmente pueden ofrecer lo mismo que se busca con la virtualización de sistemas operativos.

Una de esas técnicas es la virtualización a nivel de sistema operativo. Con un enfoque más extremo, la virtualización de sistema operativo no busca virtualizar la máquina completa sino únicamente el entorno de ejecución de usuario. Precisamente FreeBSD lleva incluyendo esta funcionalidad desde hace mucho tiempo, con los llamados FreeBSD Jails.

Lejos de toda la problemática a nivel de CPU y el resto de recursos, la virtualización de sistema operativo se centra en los mecanismos ya existentes a este nivel, tomándolos como suficientes para ofrecer el aislamiento y funcionamiento típico que se espera de un sistema operativo virtualizado. En vez de emular el hardware físico, la virtualización de SO emula otro sistema operativo usando funcionalidades del sistema operativo original.

El mecanismo se basa en limitar a ciertos procesos su acceso a parte del sistema de ficheros, bajo un directorio específico a modo de “jaula”, de tal manera que el proceso no percibe ni tiene conocimiento del resto del sistema. Si se instala un sistema operativo bajo este punto, éste podría funcionar de forma aislada, teniendo como proceso padre de todo sistema al proceso “enjaulado” desde del sistema operativo original. Añadiendo restricción a ciertas llamadas de sistema y proporcionando interfaces de red virtuales se consiguen sistemas totalmente funcionales.

A pesar de ser muy útil para ciertas aplicaciones esta aproximación nunca será tan sólida y práctica como un sistema completo virtualizado. Uno de los principales problemas es que el núcleo del sistema operativo base es compartido

entre todas las instancias virtual y si este falla por alguna razón todas las máquinas en ejecución se verán afectadas.

En el lado más positivo, no hay pérdidas por sobrecargas entre capas de virtualización ya que todo ocurre de forma independiente del hardware, y el rendimiento es de hecho, nativo.

Como alternativa para algunas aplicaciones que requieran aislamiento siempre puede ser una opción útil. Además la independencia del hardware permite que la técnica sea usada dentro de un sistema ya virtualizado, complementando su funcionalidad con más entornos independientes.

3. Virtualización con Xen

3.1. El hipervisor Xen

Xen es un proyecto de hipervisor de sistemas paravirtualizados desarrollado en la Universidad de Cambridge, presentado públicamente en el año 2003 [Barham 2003]. Aunque es comercializado actualmente por Citrix tras adquirir el proyecto, Xen es gratuito y Citrix continúa ofreciendo el código abierto. Xen es usado como base en otros productos comerciales y de libre distribución que ofrecen virtualización de servidores, infraestructuras bajo demanda (IaaS) y aplicaciones de seguridad.

El hipervisor Xen es una capa de abstracción software ubicada directamente sobre el hardware y entre todos los sistemas operativos virtualizados en un sistema. Es responsable de repartir la CPU y del particionado de la memoria de las máquinas virtuales que corren sobre el hardware. El hipervisor abstrae el hardware para las máquinas virtuales y controla su ejecución pero no realiza ninguna operación que vaya más lejos de este tipo de control de los recursos básicos. Xen no tiene ningún conocimiento de redes, almacenamiento ni nada relacionado con entrada/salida generalmente existente en el hardware. Para todas estas funciones y la gestión de las máquinas virtuales, llamadas dominios en Xen, el hipervisor delega el control a una máquina virtual privilegiada, llamada dominio cero o dominio de control.

Limitando las funciones del hipervisor a interactuar como interfaz de las máquinas virtuales su tamaño es muy reducido. Gracias a este diseño Xen es más seguro y robusto que otros hipervisores.

Xen también es independiente del sistema operativo de control. La mayoría de las implementaciones suelen usar Linux en el dominio de control pero existen también otros sistemas operativos que pueden usarse igualmente como NetBSD y OpenSolaris. FreeBSD, de momento no tiene las capacidades suficientes para actuar de dominio de control.

3.2. Arquitectura

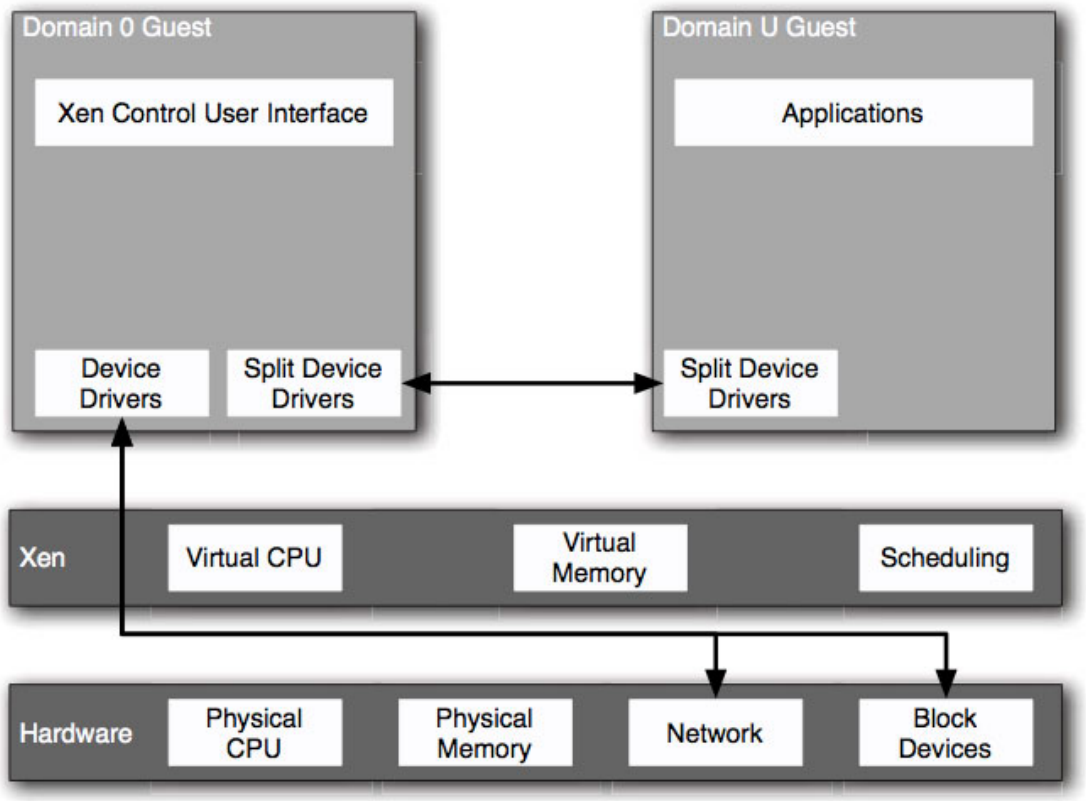

Fig. 3.1 Arquitectura típica de Xen mostrando un dominio de usuario

La arquitectura de Xen se basa en un diseño modular de componentes independientes.

El Dominio 0 o dom0 es una máquina virtual privilegiada ubicada también sobre el hipervisor Xen pero con permisos especiales de acceso a los recursos hardware como dispositivos de entrada/salida y con la capacidad de interacción con el resto de dominios virtualizados. Todas las máquinas virtuales en Xen dependen de este dominio y debe estar operativo antes de poder arrancarlas.

El Dominio 0 es el único que posee los controladores de dispositivo de red y de disco. Los dominios de usuario tiene sólo una abstracción del dispositivo y todas sus peticiones de acceso a estos recursos son dirigidas hacia el dom0, donde son gestionadas y transferidas a la red o al dispositivo de bloque de forma transparente.

Xen puede ejecutar máquinas paravirtualizadas con sistemas operativos modificados para ejecutarse sobre este hipervisor o virtualizadas por hardware con sistemas inalterados. Se les denomina generalmente domU PV y domU HVM.

Los domU paravirtualizados poseen controladores de disco y red hechos a medida que se complementan con los que están en dominio de control. En cambio en las maquinas virtualizadas por hardware no poseen drivers si no que se emplea un emulador de dispositivo de disco y red. Estos emuladores se ejecutan en el Dom0 para cada máquina HVM activa. Además los dominios virtualizados por hardware requieren emulación de BIOS.

3.2.1. Gestión de CPU

Aunque en realidad los mecanismos son algo más complejos y cada vez son más comunes los procesadores con múltiples núcleos, tradicionalmente la CPU sólo puede ejecutar una instrucción en cada momento. Las instrucciones se van encolando en pilas de ejecución y se procesan una tras otra. Independientemente del tipo de virtualización, la CPU se comparte siempre como un recurso más aplicando división en el tiempo. De esta forma las colas de instrucciones que mantiene cada máquina virtual van ejecutándose en los intervalos que les ceda el monitor de máquina virtual.

En Xen, la gestión de CPU para las máquinas virtuales es análoga a la gestión de procesos de un sistema operativo sólo que en Xen estos procesos son a su vez dominios virtuales completos. Xen gestiona los procesos de dominio virtual asignándoles CPU en función a sus propias estimaciones y demanda de cada momento.

Cada dominio virtual lleva sus propias colas de qué instrucciones instrucciones va a ejecutar a continuación pero nunca las pone directamente en la CPU bajo su control sino que notifica al hipervisor qué proceso debe ser ejecutado a continuación.

3.2.2. Gestión de memoria

El hipervisor tiene autoridad total sobre la memoria y debe controlar toda memoria asignada a los dominios. Xen maneja la memoria a nivel físico y de páginas siendo los sistemas operativos de los dominios de usuario los encargados de realizar todo el resto de operaciones de memoria.

La arquitectura x86 usa un esquema de memoria híbrido ya de por si bastante complejo. Por un lado está la memoria física de la máquina que es accedida mediante una dirección numérica. Pero esta no se usa directamente si no que se implementa una memoria virtual que permite a cada proceso acceder a memoria como si fuera el único del sistema sin colisionar ni acceder a areas donde no debe, de forma transparente. Además la memoria virtual permite acceder a un espacio mucho más grande que el disponible físicamente pudiendo cambiar partes desde el disco duro cuando se necesitan.

Al igual que la memoria física, la memoria virtual también se accede por direcciones numeradas. La relación entre las direcciones virtuales y físicas se gestiona mediante las tabla de paginación, que asocia bloques de memoria física con páginas de memoria virtual.

El mecanismo de memoria virtual se aplica incluso cuando sólo hay un sistema operativo, no tiene que ver con ninguna tecnología de virtualización.

Xen actúa también bajo este mecanismo de memoria virtual interponiendo su control en la tabla de paginación. Dado que las aplicaciones tienen que pasar por Xen para actualizar sus mapas entre memoria virtual y física, Xen puede garantizar que un dominio de usuario acceda sólo a el área de memoria que le corresponde, además de ocultar la memoria del resto de dominios de su percepción.

En la arquitectura x86 la paginación se realiza parcialmente en el hardware en el MMU o Memory Management Unit del procesador. Pero además, aunque la paginación sea suficiente para garantizar la protección de memoria y la ilusión de direcciones virtuales contiguas, la arquitectura x86 emplea también segmentación para proteger el acceso a memoria y aumentar el tamaño de la memoria direccionable. Las aplicaciones no emplean directamente direcciones físicas sino lógicas, mediante un selector de segmento de 16bits y una dirección de desplazamiento respecto al selector, que luego es traducido a la posición de memoria física por el procesador.

En la práctica, los sistemas modernos intentan evitar el uso registros de segmentos y asignan directamente todo el espacio de memoria. Sin embargo este sistema de segmentación si se aprovecha en Xen para proteger el área de memoria donde reside el propio hipervisor. Xen se reserva una pequeña cantidad de memoria en las direcciones bajas de todos los dominios que crea y controla sus segmentos de forma que no incluyan este área. Además cada segmento puede tener privilegios asociados con lo que el procesador puede controlar cualquier intento de acceso al área protegida del principio del segmento.

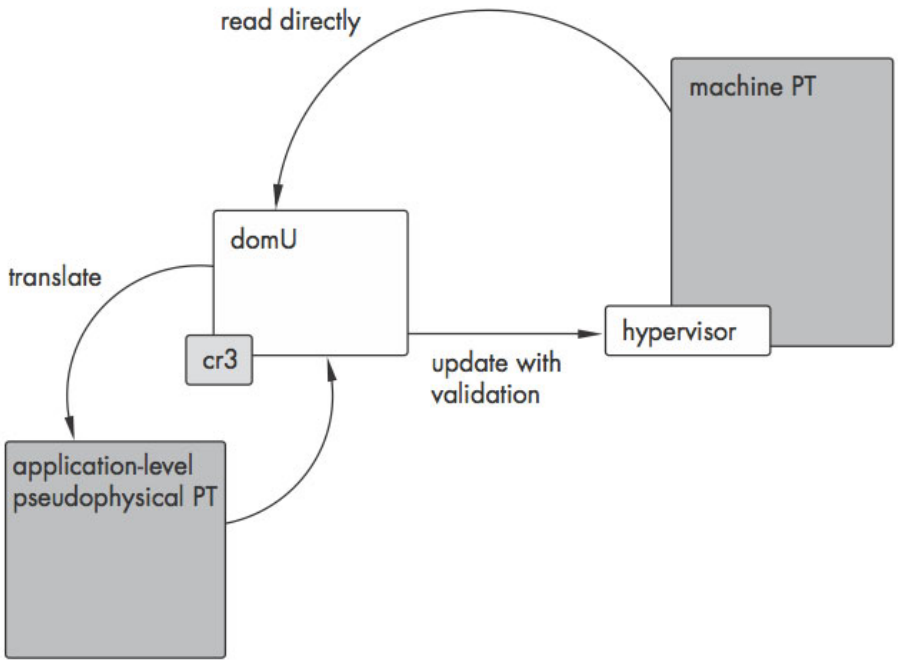

Fig. 3.2 Acceso a memoria en Xen

Finalmente, Xen añade una capa más a la gestión de memoria. Dado que la memoria física reservada para cada dominio puede estar fragmentada en distintas áreas y los sistemas operativos esperan manejar un espacio contiguo de direcciones físicas, Xen mantiene un mapa de memoria más que relaciona la memoria física con la memoria virtual que maneja el dominio de usuario. Los dominios pueden acceder directamente a este área pero cada vez que necesiten actualizar alguna página de memoria, será gestionado a través del hipervisor.

3.2.3. Interrupciones

Las interrupciones ocurren normalmente cuando algún elemento del hardware requiere atención de un controlador software. Tradicionalmente las peticiones de interrupción se tratan inmediatamente pasando el control al programa asociado y el resto de procesos esperan a que termine. En general es un método efectivo pero en el contexto de la virtualización este mecanismo no es aceptable.

En Xen todas la interrupciones son interceptadas por el hipervisor y no se pasan directamente a ningún dominio ni controlador. De esta manera Xen conserva el control del hardware en todo momento. Los dominios deben registrar primero con el hipervisor qué interrupciones necesitan y sus controladores asociados. Cuando ocurre una interrupción registrada Xen notifica al dominio asociado y le cede paso para el siguiente tiempo de ejecución. Si se producen varias interrupciones mientas un dominio está inactivo, éstas se acumulan hasta que entre en ejecución, evitando cambios continuos de contexto y mejorando el rendimiento de Xen.

3.2.4. Entrada/Salida y dispositivos

Los dominios de usuario no tienen permiso para manejar dispositivos por sí mismos. El modelo de Xen debe garantizar la seguridad para que no pueda haber ningún tipo de interferencia o acceso desde un dominio de usuario y el hardware u otros dominios. Todo acceso debe ser gestionado a través del hipervisor, con la ayuda del dominio de control Dom0.

Xen maneja la entrada y salida de los dominios usando canales de dispositivo y dispositivos virtuales. Estos no son más que enlaces punto a punto entre el dispositivo ubicado en el dom0 y el dispositivo del domU, para los cuales se implementan buffers circulares donde se escriben por un lado datos para ser leídos y en el extremo opuesto se van extrayendo una vez atendidos.

Xen aplica este mecanismo a los dispositivos de red y de bloques.

Por un lado la arquitectura de red esta pensada para ser lo más sencilla posible. Xen proporciona interfaces de red virtuales a los dominios a través de canales de dispositivo funcionando como un medio donde los paquetes de las interfaces virtuales de los dominios llegan a directamente otro interfaz virtual en el dominio de control. Cualquier otra funcionalidad se deja a las utilidades de red estándar.

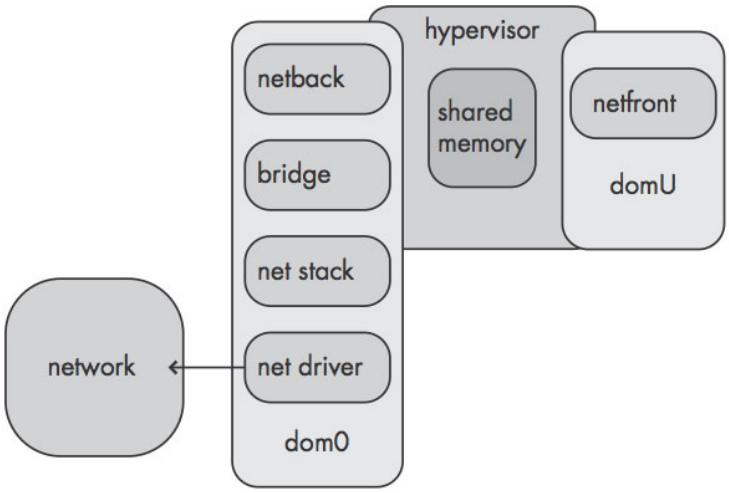

Fig 3.3 Pila de red en Xen

Externamente, hipervisor actua únicamente como canal de datos de forma que los paquete puedan moverse desde el interfaz físico hacia el interfaz virtual del dominio. El control y validación de los paquetes se deja al Dominio 0 mediante reglas definidas en iptables. Al emplear herramientas estándar unix, Xen deja mucho margen a la configuración de red.

En cuanto al interfaz de red virtual en sí es algo relativamente simple. Básicamente es un buffer de memoria donde escribir paquetes, otro donde leer y llamadas al hipervisor para notificar nuevos datos.

Los dispositivos de bloque funcionan de una forma muy parecida a los de red. El hipervisor proporciona dispositivos de bloque virtuales a los dominios y deja toda la gestión al dom0, el cual provee controladores que replican la funcionalidad de un dispositivo real.

4. FreeBSD sobre Xen

4.1. El sistema operativo FreeBSD

FreeBSD es un sistema operativo UNIX de libre distribución ampliamente usado en sistemas que gestionan servicios de red. Está presente en multitud de sistemas dedicados y embebidos, hardware de conmutación de red y en general en gran cantidad de sistemas que requieran una disponibilidad mayor que los sistemas de consumo.

Creado en 1979 en la Universidad de Berkeley, California, acumula ya más de tres décadas de continuo desarrollo y refinamiento. Originalmente el sistema se llamaba BSD e incluía código licenciado por el operador americano AT&T. En 1992 el código se liberó para el público general bajo la licencia BSD or Berkeley Software Distribution que no limita de ninguna forma la reproducción, alteración o uso comercial.

Siendo otro UNIX, FreeBSD es muy distinto al resto de sistemas operativos en aspectos que van más allá de lo técnico o lo funcional. A parte de su gran estabilidad, estos son algunos de sus puntos fuertes:

Portabilidad

El objetivo del proyecto FreeBSD es proveer un sistema operativo de libre distribución estable, seguro y que funcione sobre el hardware más común y accesible para el público. Hoy en día se trata de sistemas compatibles con la arquitectura Intel x86 y más recientemente, amd64 para sistemas de 64 bits. Anteriormente existían muchas más arquitecturas en entornos que podían sacar provecho de FreeBSD como los sistemas SPARC, Itanium y PowerPC pero han caído en desuso y aunque continúan soportadas, son de menor categoría. ARM, para entornos embebidos, es una de las nuevas arquitecturas soportadas por FreeBSD con más potencial.

Potencia

FreeBSD no necesita un sistema fuera de lo común para funcionar. Funciona extremadamente bien en sistemas modernos de cualquier tipo de capacidad, desde el más básico x86. Sus mínimos requerimientos y su mínimo impacto permiten dedicar al máximo los recursos hardware a la tarea para que se los necesite. FreeBSD también soporta sistemas multi CPU y multicore más recientes.

Gestión sencilla

FreeBSD se gestiona y configura de forma realmente simple y coherente. Incluye herramientas y mecanismos como la colección de Ports que simplifican enormemente en la tarea de configurar nuevo software automatizando y documentando la instalación, desinstalación y configuración de procesos para miles de paquetes de software.

Soporte de upgrades

Al contrario de otros sistemas que requieren procedimientos de upgrade arriesgados y complicados, FreeBSD permite realizar upgrades sencillos generando un nuevo sistema optimizado para un determinado hardware y aplicaciones. De esta forma se aprovechan todas las funcionalidades soportadas por un hardware en vez del menor denominador común.

Sistema de archivos avanzado

FreeBSD soporta multitud de sistemas de archivos, algunos muy sofisticados. Su sistema de archivos por defecto es altamente resistente a daños y ofrece una latencia excelente de lectura y escritura. De hecho, el sistema de archivos BSD es tan avanzado que otros sistemas UNIX comerciales lo han usado como base de su propio sistema de archivos.

4.2. Soporte Xen en FreeBSD

La paravirtualización, inicialmente, y a diferencia de la virtualización total, requiere una serie de modificaciones en el funcionamiento interno del sistema operativo que lo hacen consciente y cooperativo con el monitor de virtualización o hipervisor sobre el que se apoya a modo de nueva capa, compartiendo los recursos con otras máquinas virtuales.

Mientras que Linux ha sido el principal sistema operativo portado a Xen y existiendo incluso versiones modificadas de Windows, FreeBSD ha mostrado un ritmo muy lento en su integración sobre el hipervisor Xen. Desde la aparición de Xen, el código de FreeBSD ha ido incluyendo aportaciones que permiten su ejecución paravirtualizada sobre el hipervisor. Unas veces etiquetadas de experimentales, otras veces semifuncionales, estas opciones limitadas se alejaban de la robustez y compatibilidad normalmente asociadas a FreeBSD.

Con todo, las tecnologías de virtualización tampoco han permanecido inmóviles. En los últimos años se han ido refinando y generalizando modos de virtualización asistidos por hardware, que posteriormente han girado hacia modalidades híbridas. Durante esta evolución los intentos de hacer FreeBSD compatible con Xen también han ido cambiado el foco entre un modo y otro, buscando sacar provecho de implementaciones con mejor rendimiento o con menor impacto en el código del sistema.

Con la virtualización asistida por hardware, la nueva versión 10 de FreeBSD liberada en Enero de 2014 ofrece la mejor integración con Xen hasta la fecha. Los notables cambios y mejores hacen al fin de Xen una opción viable de virtualización para FreeBSD.[2]

[2] http://blog.xen.org/index.php/2014/01/21/improved-xen-support-in-freebsd/

4.3. Modos de virtualización Xen en FreeBSD

Actualmente FreeBSD mantiene soporte para dos arquitecturas hardware: i386 y amd64, correspondientes a la arquitectura de 32 bits de intel IA-32 de la familia x86 y a la arquitectura AMD64 de 64 bits, compatible también con la equivalente de intel. La integración con Xen requiere una implementación distinta para cada una de las plataformas. Por esta razón existe una diversidad entre los modos de Xen soportados para cada una de ellas.

En principio Xen sólo estuvo disponible de forma experimental en la arquitectura de 32 bits en modo paravirtualizado y actualmente sigue limitado a esta arquitectura.

Con el soporte de virtualización hardware es posible correr cualquier versión de FreeBSD en Xen aunque sólo las amd64 ofrecen controladores de disco y red paravirtualizados.

En la última versión se ha introducido soporte Xen mediante virtualización por hardware híbrida que incluye funciones del sistema paravirtualizadas como las interrupciones o el control integrado de dispositivos, además de los propios controladores de red y bloque. En cuanto al soporte de paravirtualización híbrida, aún se trata de un modo experimental que tampoco esta finalizado en Xen o XenServer por lo que no será no se incluirá en el ámbito del estudio práctico.

FreeBSD	**6.x**	**7.x**	**8.x**	**9.2**	**10**
HVM	**i386, amd64**	**i386, amd64**	**i386, amd64**	**i386, amd64**	i386, amd64
HVM + Drivers Xen	-	-	amd64	amd64	-
PVHVM	-	-	-	-	**amd64, i386**
PVH	-	-	-	-	Experimental amd64
PV	Experimental i368	Experimental i386	i386	i386	i386

Tabla 4. Modos de virtualización Xen soportados por versión de FreeBSD

4.3.1. FreeBSD paravirtualizado (PV)

Fue la primera aproximación existente de FreeBSD sobre Xen. El soporte oficial a Xen apareció en la versión FreeBSD 8.2. Puede consultarse directamente en el manual ejecutando “man xen” [3]

Para poder obtener el modo paravirtualizado, FreeBSD requiere recompilar todo el núcleo. Existe una configuración de núcleo predefinida denominada XEN que puede usarse directamente para compilar.

El núcleo paravirtualizado incluye toda la integración con los dispositivos Xen e incluye por defecto el controlador dispositivo de red virtual y de bloque.

Hasta la versión 10 no se soportaba migración ni el cliclo completo de máquina virtual Xen que incluye suspensión, estado que es requerido para poder realizar la migración en caliente.

Actualmente las mayores limitaciones que presenta este modo es que está limitado a la propia arquitectura de 32 bits no pudiendo asignare más de 4GB de memoria. Además no soporta multicore o multiCPU ya que existe un fallo en el código que aún no se ha corregido por lo que no se pueden asignar más de una vCPU a las instancias FreeBSD virtualizadas con este modo.

En general el código de este modo no está tan probado y refinado como lo están las versiones tradicionales de FreeBSD por lo que existe cierto riesgo o pueden encontrarse otras limitaciones no documentadas.

[3] http://www.freebsd.org/cgi/man.cgi?query=xen&manpath=FreeBSD+8.2-RELEASE

4.3.2. FreeBSD virtualizado por hardware (HVM)

Este modo es básicamente cualquier FreeBSD no modificado.

Técnicamente es el más fiable ya que se trata de las mismas versiones de producción extensivamente probadas.

Por contra el no tener ninguna integración con Xen hace que tenga peor rendimiento. Especialmente en los dispositivos de entrada salida los cuales están emulados en el dominio de control.

A partir de la versión 10, el modo por defecto es PVHVM por lo que si quiere funcionarse sin ningún soporte Xen el núcleo ha de ser recopilado eliminando toda compatibilidad Xen del kernel GENERIC.

4.3.3. FreeBSD virtualizado por hardware con drivers Xen (HVMXEN)

Esta versión funciona básicamente como el modo HVM comentado anteriormente pero incluye dentro del kernel el bus con dispositivos Xen. Estos controladores no están disponibles como software separado y debe recopilarse el núcleo para poder obtener su funcionalidad.

El soporte de este modo está limitado a la arquitectura amd64 y está disponible con la configuración de kernel denominada HVMXEN.

En la versión actual de FreeBSD 10 ya no hay posibilidad de obtener esta configuración debido a que esta nueva versión incluye el soporte Xen en el kernel genérico y no es posible desactivar parte de su funcionalidad. Esta forma queda ya limitada a las versiones anteriores de FreeBSD.

4.3.4. FreeBSD con virtualización híbrida (PVHVM)

Este es el modo más avanzado actualmente aunque no se trata de un modo paravirtualizado ya que depende en gran parte de funciones virtualizadas por hardware.

Esta versión incluye la integración con el bus Xen incluyendo la gestión de interrupciones paravirtualizada, dispositivos auxiliares y los dispositivos de red y bloque..

FreeBSD funciona en este modo por defecto desde la versión 10.

5. Estudio del rendimiento de FreeBSD sobre Xen

5.1. Objetivos

El principal objetivo de este estudio es conocer el rendimiento de todos los tipos de versiones FreeBSD que pueden ejecutarse sobre Xen. Dado que existen múltiples combinaciones entre tipo de virtualización y arquitectura, se acotará el estudio a las más representativas. Se ha tomando como base la versión más reciente del sistema, FreeBSD 10 con la excepción del modo HVM optimizado con divers Xen (HVMXEN), sólo disponible en la versión anterior, 9.2.

Como arquitectura de referencia no es posible tomar un mínimo denominador común. Por un lado la versión paravirtualizada sólo está disponible para i386. Por el contrario la versión HVMXEN sólo está en amd64. En cambio la versión PHVM sí soporta ambas arquitecturas. Se han decidido incluir las dos variantes, a modo de versión de control, siendo además la más reciente y avanzada.

Las versiones objeto de estudio quedan acotadas al siguiente conjunto:

FreeBSD 10 i386 PHVM
FreeBSD 10 i386 PV
FreeBSD 10 i386 HVM
FreeBSD 9.2 amd 64 HVMXEN
FreeBSD 10 amd64 PHVM

5.2. Aspectos de estudio

Los entornos virtualizados han sido objeto de numerosos estudios y publicaciones con los mismos objetivos de medir y modelar el rendimiento de una máquina virtual determinada.

Huber [Huber 2010] identifica los factores de la virtualización que influyen en el rendimiento. Estos se dividen principalmente en el tipo de virtualización, la gestión de los recursos y el tipo de carga asociado al sistema. (Fig 5.1).

El tipo de carga de un sistema distingue entre e CPU, memoria y la entrada salida por disco y red como factores de rendimiento de una plataforma virtual. En cada tipo de plataforma, estos recursos se virtualizan de formas muy distintas. Un sistema con carga intensiva alguno de en estos factores obtendrá resultados muy distintos en cada plataforma virtual.

Para este estudio se tomarán como indicadores de rendimiento los resultados de pruebas centradas en estos cuatro factores: carga de CPU, acceso a memoria, acceso a disco y red.

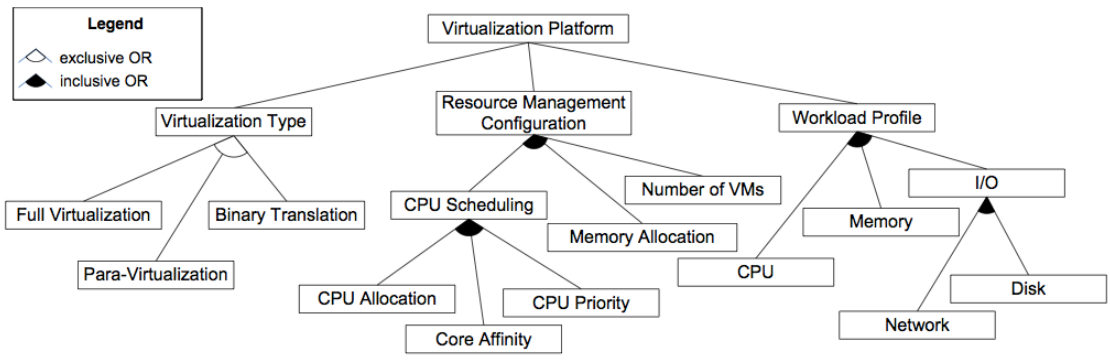

Fig. 5.1 Factores influyentes en el rendimiento de un sistema virtualizado.

6. Sistema de pruebas

6.1. Diseño e implementación

El sistema de pruebas consiste principalmente en un servidor dedicado sobre el que se ejecutan las distintas máquinas virtuales y un sistema auxiliar que lleva el control de los tests. Ambas máquinas están conectadas punto a punto mediante interfaces Ethernet gigabit.

En el servidor, se instaló la última versión de Citrix Xen Server disponible. Una vez configurado el hipervisor, se instalaron manualmente cada una de las versiones de FreeBSD Xen objeto de estudio.

Las imágenes de disco virtual se ubican de forma local en el disco duro del servidor. Por simplicidad y restar incertidumbre a los resultados, se descartaron otras implementaciones más flexibles que permite Xen Server como el almacenamiento en unidades de red NFS o iSCSC.

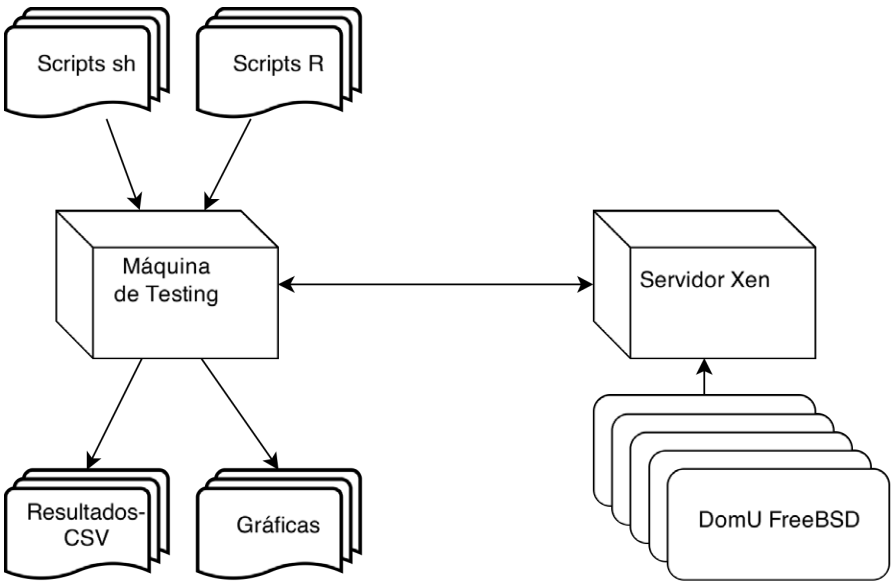

Fig. 6.1. Esquema simplificado del sistema de pruebas.

En la creación de los distintos DomUs se ha intentado desviarse lo mínimo posible de una instalación estándar de FreeBSD. En todos los casos se empleo el particionado generico, se habilitó el acceso por ssh y se instalaron mediante paquetes binarios las herramientas necesarias para los propios tests.

Para los modos Xen que requieren recompilar el kernel FreeBSD se utilizaron los ficheros de configuración originales denominados XEN, XENHVM y GENERIC. Para el kernel XEN se deshabitaron las opciones de debugging WITNESS e INVARIANTS, que afectan rendimiento final. Para el modo total mente asistido por hardware (HVM) se desactivaron de la configuración GENERIC todas las opciones de soporte Xen que FreeBSD 10 incluye ahora por defecto.

En el sistema auxiliar se ejecuta el programa de control, se copian los resultados y también se procesan. El control de cada test se basa en scripts UNIX sh que incluyen comandos ssh dirigidos al servidor Xen y a las máquinas virtuales FreeBSD. Los resultados generados en cada prueba se guardan en ficheros de texto plano separado por comas, siguiendo el formato CSV. Finalmente estos ficheros son procesados por sencillos scripts en lenguaje R capaces de generar gráficas y estadísticas.

Modelo	Mac Mini i5 MD387LL/A “off the self”
CPU	Intel i5 2.5 GHz con soporte VTx y EPT
RAM	4 GB 1600 MHz DDR3 SDRAM
Disco	SATA 500GB 5400rpm 2’5”
Red	Ethernet Gigabit
SO	Citrix XenServer 6.2.0-70446c

Tabla 6.1. Características del servidor de pruebas.

SO	Mac OS X 10.9.4
Herramientas	bash, sh, R 3.0.3, iperf 2.0.5
Red	Ethernet Gigabit

Tabla 6.2. Características relevantes del sistema auxiliar de pruebas.

vCPU	1
Memoria	256MB
Disco	8GB
Disco para pruebas	2GB

Tabla 6.3. Recursos asignados los dominios Xen FreeBSD

6.2. Herramienta de pruebas automáticas

Para la realización de las pruebas se optó por desarrollar totalmente a medida una herramienta programada en sell script.

Por un lado, el control automatizado basado en test programados garantiza que los resultados sean coherentes al ser obtenidos de la misma forma en todos los sistemas objeto de test. Por otro lado, la automatización permite controlar el aislamiento de cada máquina virtual, mediante el apagado y encendido sincronizado de las máquinas virtuales. Cada prueba se realiza sobre una máquina virtual recién iniciada y con ejecución exclusiva sobre el hipervisor. Se garantiza así la no interferencia por los recursos hardware y se aumenta la fiabilidad de los resultados. Las pruebas de escalabilidad o que impliquen tener más de una máquina en ejecución en el hipervisor, quedan fuera de los objetivos de este trabajo.

Las pruebas se ejecutan de forma secuencial para cada máquina virtual y son repetidas varias veces para disponer de un número suficiente de muestras.

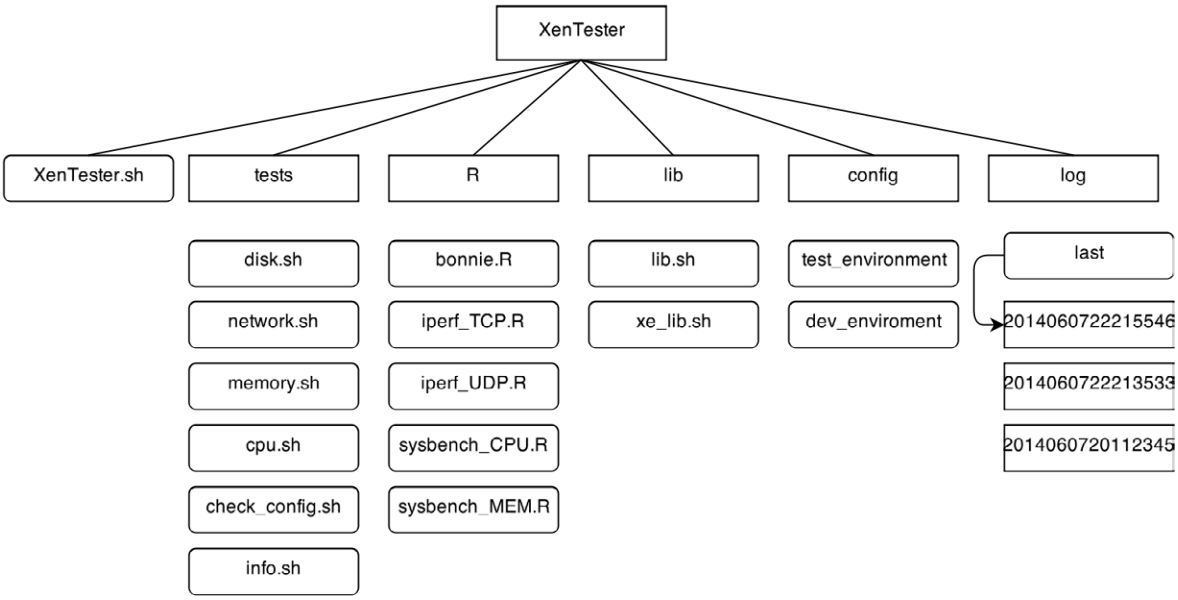

Fig. 6.2. Estructura de ficheros y directorios de la herramienta de pruebas

El programa principal contiene una lógica basada en bucles y su comportamiento depende del numero de objetos que carga en el arranque: las máquinas virtuales conocidas y los ficheros con las pruebas.

Al arrancar, obtiene como parámetro de entrada el fichero con la configuración del entorno de pruebas. Este fichero incluye la IP del Dom0 y cada una de las DomU FreeBSD instaladas. Toda la comunicación y control de las máquinas remotas se realiza mediante llamadas ssh. Para poder ejecutar comandos de forma remota con autenticación automática, se han intercambiado previamente las claves públicas ssh entre la máquina de pruebas, el Dom0 y todos los Dom0 FreeBSD.

Las pruebas programadas se leen directamente del directorio “tests” y son independientes entre sí. Esto proporciona gran flexibilidad y extensibilidad a la herramienta. Se pueden añadir o quitar pruebas con sólo copiar ficheros a esta carpeta.

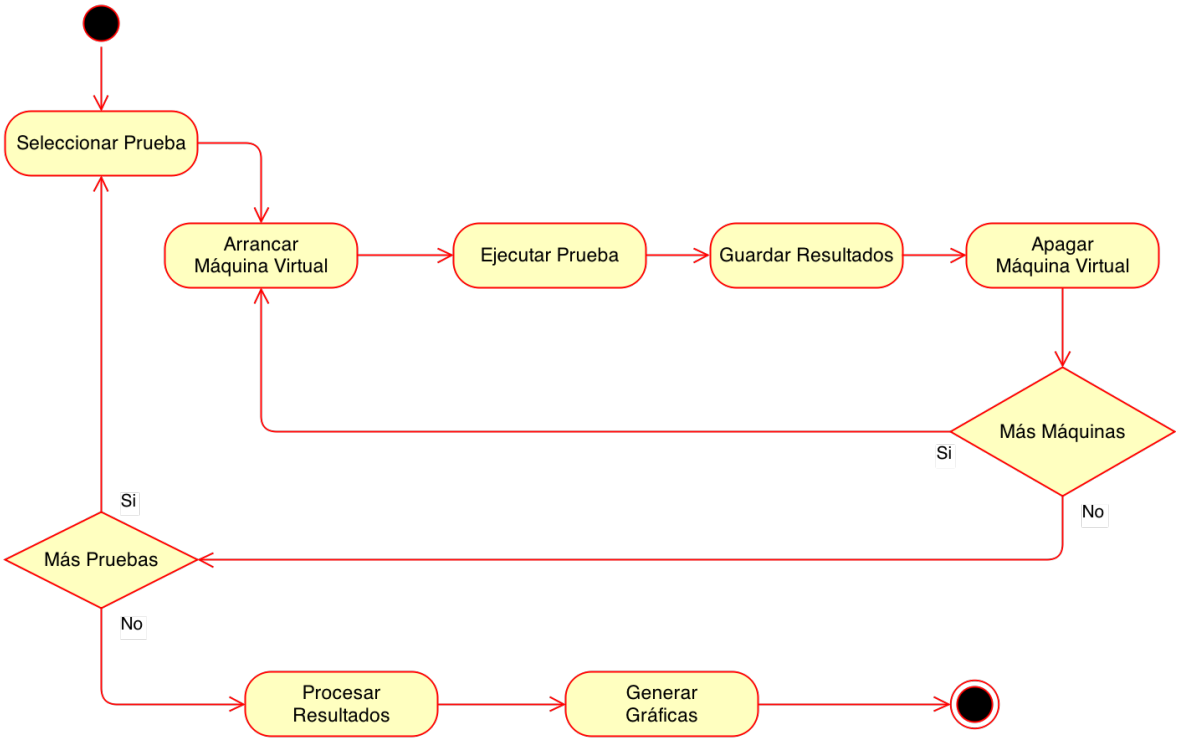

Fig. 6.3. Lógica de control de la herramienta de pruebas.

Los scripts de pruebas están divididos en dos partes. La parte principal contiene la prueba en sí y básicamente son llamadas a las herramientas de benchmarking instaladas en las máquinas virtuales. La segunda parte, contiene órdenes para procesar la información extraída de la prueba.

Para cada test y para cada máquina virtual se recoge información en ficheros independientes. Al final de la prueba se procede a unificar los resultados en un mismo fichero de texto plano en formato CSV. Los ficheros CSV contienen datos de forma tabular comenzando por una cabecera descriptiva en la primera línea y en el resto los valores obtenidos en cada muestra incluyendo además el nombre de máquina al que corresponden. Este identificador será el que se utilice al final del proceso para clasificar las muestras obtenidas de forma gráfica.

```
timestamp,source_ip,source_port,dest_ip,dest_port,tag,interval,transferred_bytes,bits_per_second,name
20140830025356,192.168.10.22,38907,192.168.10.1,5001,3,0.0-1.0,9937200,79497600,FreeBSD10-i386-HVM
20140830025357,192.168.10.22,38907,192.168.10.1,5001,3,1.0-2.0,11326350,90610800,FreeBSD10-i386-HVM
20140830025358,192.168.10.22,38907,192.168.10.1,5001,3,2.0-3.0,10379670,83037360,FreeBSD10-i386-HVM
20140830025359,192.168.10.22,38907,192.168.10.1,5001,3,3.0-4.0,10766280,86130240,FreeBSD10-i386-HVM
20140830025400,192.168.10.22,38907,192.168.10.1,5001,3,4.0-5.0,10722180,85777440,FreeBSD10-i386-HVM
20140830025400,192.168.10.22,38907,192.168.10.1,5001,3,0.0-5.0,53133150,85010336,FreeBSD10-i386-HVM
…
```

Fig. 6.4. Extracto de fichero CSV con resultados del test de red UDP

El procesado de datos y la generación de gráficas también se realiza con scripts independientes. Una vez que han finalizado todas las pruebas, se llaman de nuevo a los scripts de pruebas para ejecutar la parte de procesado que incluye cada uno de ellos. En esta parte se realiza una llamada a un tercer script, escrito en lenguaje R, que genera como salida gráficas en ficheros de imagen.

7. Resultados

En las tablas se resumen las versiones de FreeBSD objeto del estudio y la nomenclatura empleada para referirse a cada una de ellas. A excepción de la máquina virtualidad por hardware con drivers Xen, que sólo está disponible en FreeBSD 9.2 y arquitectura amd64, todas las versiones son FreeBSD 10 i386.

Excepcionalmente, se han realizado pruebas extra comparando las versiones de FreeBSD 10 en modo híbrido para las versiones de 64 y 32 bits, con el objeto de conocer también el rendimiento para la arquitectura amd64 del modo más evolucionado.

Referencia	Versión FreeBSD	Arq.	Tipo Virtualización	Configuración Kernel
FreeBSD 10 i386 PVHVM	10	i386	PVHVM	GENERIC
FreeBSD 10 i386 PV	10	i386	PV	XEN
FreeBSD 10 i386 HVM	10	i386	HVM	GENERIC*
FreeBSD 9.2 amd64 HVMXEN	9.2	amd64	HVMXEN	HVMXEN

Tabla 7.1 Versiones de FreeBSD objeto del estudio.

Referencia	Versión FreeBSD	Arq.	Tipo Virtualización	Configuración Kernel
FreeBSD 10 i386 PVHVM	10	i386	PVHVM	GENERIC
FreeBSD 10 i386 PVHVM	10	amd64	PVHVM	GENERIC

Tabla 7.2 Versiones de FreeBSD comparadas en algunas pruebas.

Tipo de virtualización	Referencia
Paravirtualización	PV
Virtualización asistida por HW + Drivers Xen + Paravirtualización Interrupciones	PVHVM
Virtualización asistida por hardware + Drivers Xen	HVMXEN
Virtualización asistida por hardware	HVM

Tabla 7.3 Nomenclatura para tipo de virtualización

7.1. CPU

El test de CPU tiene como objetivo cuantizar el rendimiento que el sistema operativo es capaz de obtener del procesador.

Debido a la limitación de la versión de FreeBSD i386 paravirtualizado para funcionar en modos multiprocesador todos los DomU FreeBSD del entorno de pruebas han sido configurados con una única CPU virtual o vCPU. Este mínimo denominador común es además el escenario más sencillo posible. Como ventaja para la comparativa, permite sacar de la ecuación todas las diferencias debidas a distintas implementaciones de gestión multi CPU que pudieran tener cada uno de los sistemas FreeBSD. No obstante es muy probable que fueran mínimas puesto que se trata del mismo sistema operativo con ciertas variaciones.

Para la medición se empleó la herramienta de benchmarking *sysbench*, disponible como paquete binario y port de FreeBSD. En el modo de medida CPU la herramienta realiza el calculo de números primos hasta un máximo dado.

Se realizaron 30 repeticiones del test para el cálculo hasta el número 25000:

```
sysbench --test=cpu --cpu-max-prime=25000
```

Como puede apreciarse en la gráfica, las muestras son consistentes y apenas sufren variaciones para cada máquina virtual. Entre las distintas máquinas la diferencia también es mínima.

Reduciendo las muestras al valor medio se obtiene un resultado más claro del escenario de prueba. Un tiempo menor indica un mejor rendimiento en la prueba.

Como puede apreciarse la versión paravirtualizada obtiene una ligera ventaja sobre las implementaciones, a excepción de la versión con drivers xen HVMXEN. Sin embargo este segundo hecho no es del todo válido.

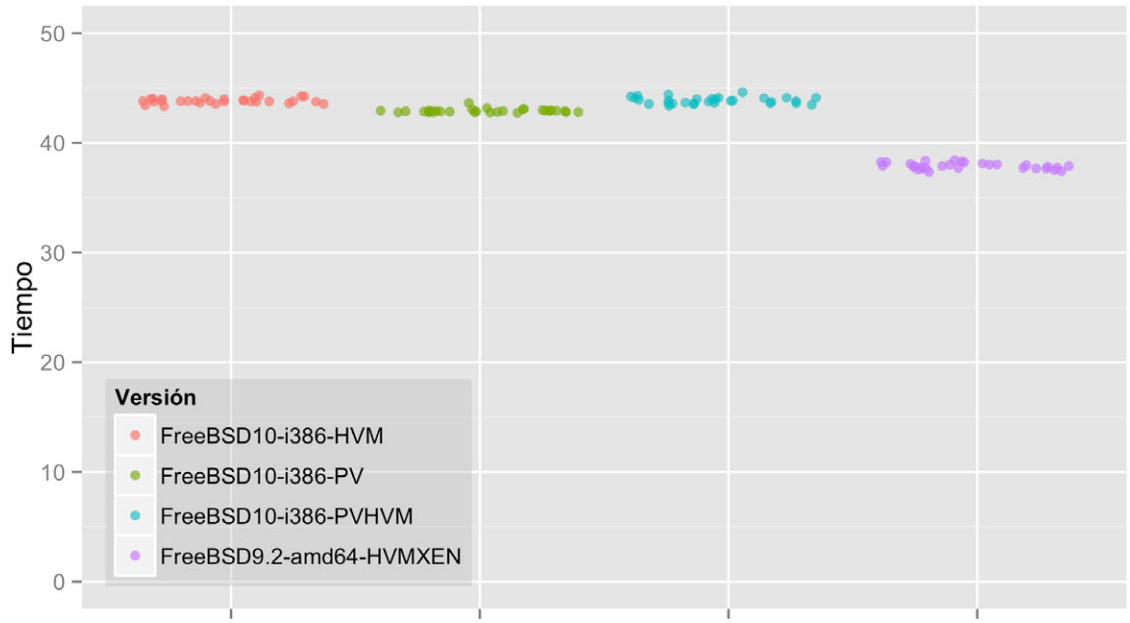

Fig. 7.1. Muestras obtenidas en el test de CPU

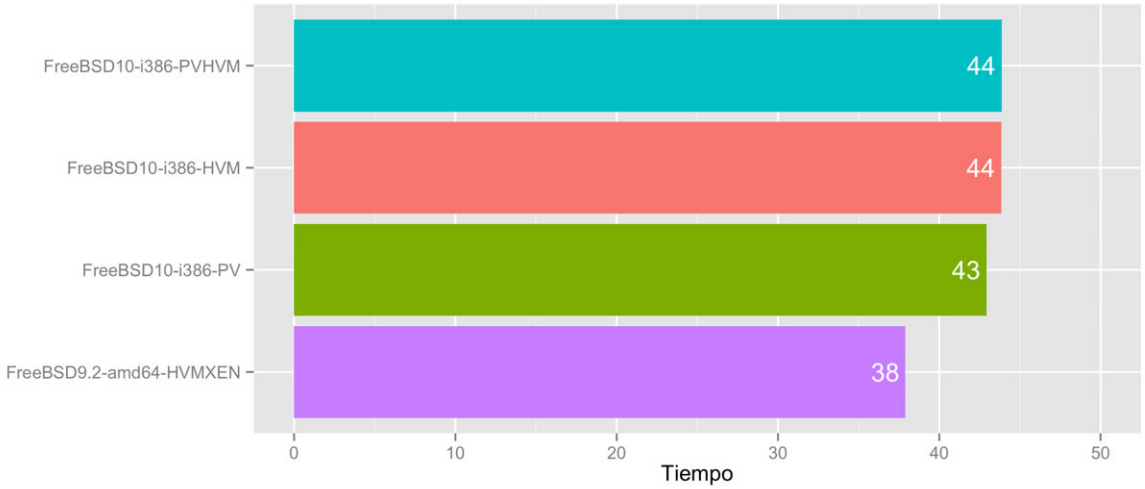

Fig. 7.2. Comparativa en valor medio para test de CPU

Recordemos que la versión HVMXEN no esta disponible en FreeBSD 10 y que en la versión anterior 9.2 este modo sólo está para la arquitectura amd64. Esta versión se ha incluido en todos los escenarios de test con el objetivo de poder comparar resultados de todas las implementaciones de FreeBSD disponibles. La versión con divers Xen debe ofrecer mejoras en rendimientos de operaciones de entrada/salida. Sin embargo en el caso de la CPU el hecho de usar una versión de 64 bits frente a otras de 32bits no permite una comparativa adecuada.

Si obviamos esta versión de 64 bits y nos centramos únicamente en las de 32 podemos concluir que aunque el modo paravirtualizado muestra una pequeña ventaja, no hay diferencias sustanciales entre las distintas versiones de FreeBSD sobre Xen en cuanto a rendimiento de CPU.

Como prueba extra se ha incluido una comparativa entre resultados amd64 e i386 para FreeBSD 10 PVHVM aunque no forme parte de los objetivos comparar entre arquitecturas. Como puede verse en la gráfica el rendimiento para el test de CPU es mejor en la versión de 64bits y este es además muy cercano al que se obtuvo para FreeBSD 9.2 amd64 HVM. Esto refuerza la conclusión final de que la CPU como recurso virtualizado no sufre especial penalización entre los distintos modos de virtualización.

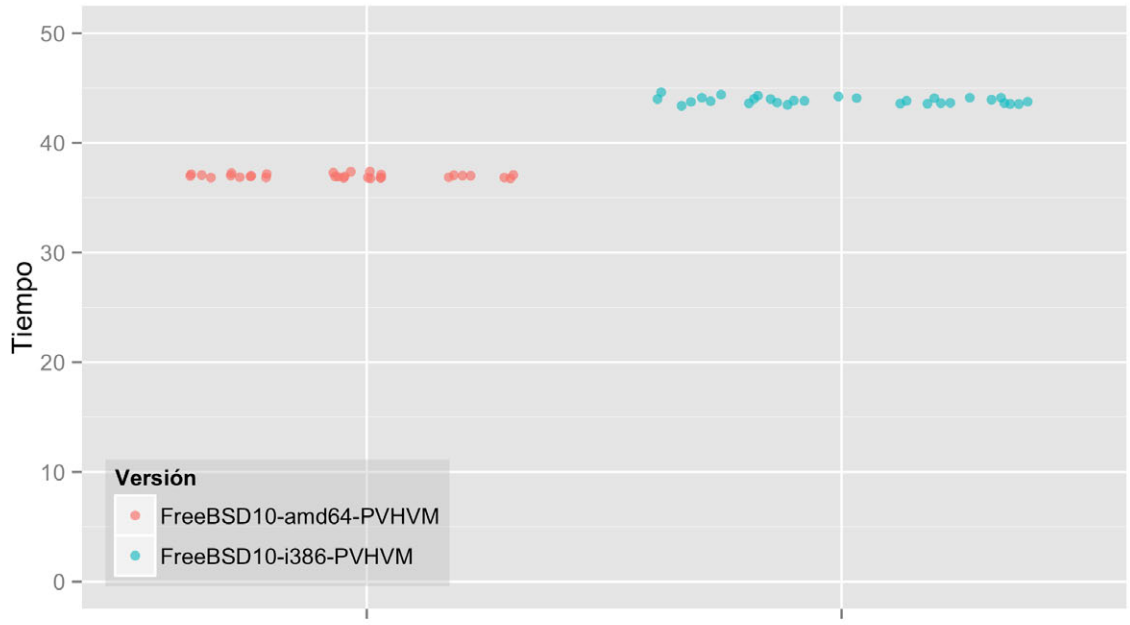

Fig. 7.3. Resultados del test de CPU para PVHVM amd64 vs i386

7.2. Memoria

Las pruebas de rendimiento de acceso a memoria se realizaron también mediante la herramienta de benchmarking *sysbench.*

El test de memoria consiste en la escritura de bloques con un volumen total alto. Se mide el tiempo que se tarda en escribir hasta el último bloque. Además se la herramienta permite realizar el test con un numero dado de hilos de ejecución de forma que la escritura se distribuya de manera simultánea entre ellos.

Se realizaron 30 repeticiones para ejecuciones con 1, 2, 4, 8, 16, 32 y 64 hilos de ejecución. En todos los casos se tomaron bloques de 16K para un volumen total de 128MB. La cantidad escrita no es especialmente alta pero este valor está limitado por la memoria virtual asignada a las máquinas, 256MB.

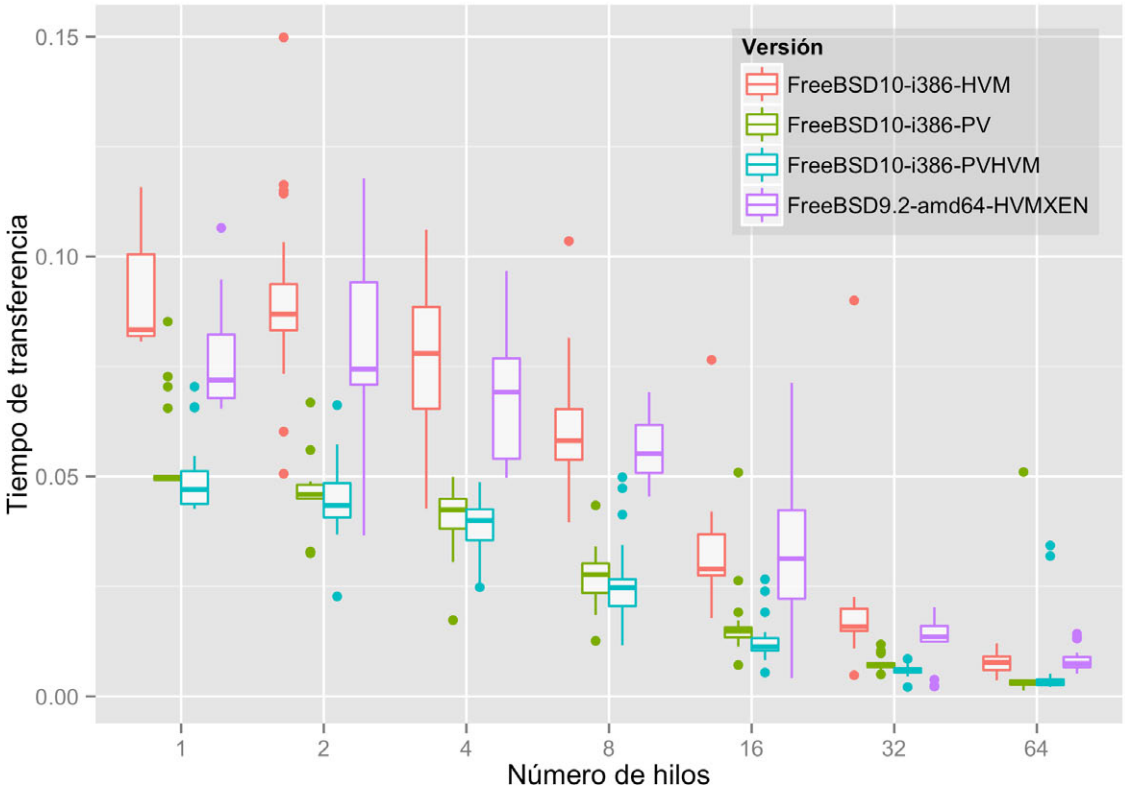

Fig. 7.4. Muestras obtenidas para el test de acceso a memoria

Aunque es posible modificar la cantidad de memoria virtual asignada antes de arrancar una máquina virtual Xen, se opto por mantenerlo consistente a lo largo de todas las pruebas y no se cambió a otro mayor para esta. En realidad, el único parámetro relevante para la prueba es la velocidad de escritura total del test y no depende del tamaño total escrito.

```
sysbench --test=memory --memory-block-size=16K --memory-scope=global --
memory-total-size=128M --memory-oper=write –num-threads=$num_thread run
```

Las muestras extraídas del test son algo aleatorias como para tomarlas directamente en consideración. Para resumir este comportamiento y reducirlo a valores más estables se han representado mediante diagramas de cajas. Para máquina y cada prueba en función de los hilos seleccionados.

En todos los casos, la implementación PVHMV obtiene ventaja sobre las demás. Le sigue muy de cerca la implementación paravirtualizada pura. Las versiones HVMXEN y HVM, menos optimizadas, no solo muestran unos valores medios mucho peores sino que además como puede apreciarse en el tamaño de las cajas y los bigotes, son mucho menos precisas en la ejecución.

En cuanto a las diferencias entre PVHVM i386 y amd64, si mostraba el mejor rendimiento frente a las otras implementaciones, la arquitectura de 64 bits ofrece unos resultados aún mejores.

Puede concluirse que en lo relativo a acceso a memoria, la paravirtualización híbrida obtiene un rendimiento superior a todas las implementaciones. A diferencia de la paravirtualización clásica, este modo híbrido hace uso del hardware dedicado de acceso a memoria. En esta prueba ha quedado demostrado que gracias al avance en soporte hardware para la virtualización, la paravirtualización clásica no es siempre la opción más optima.

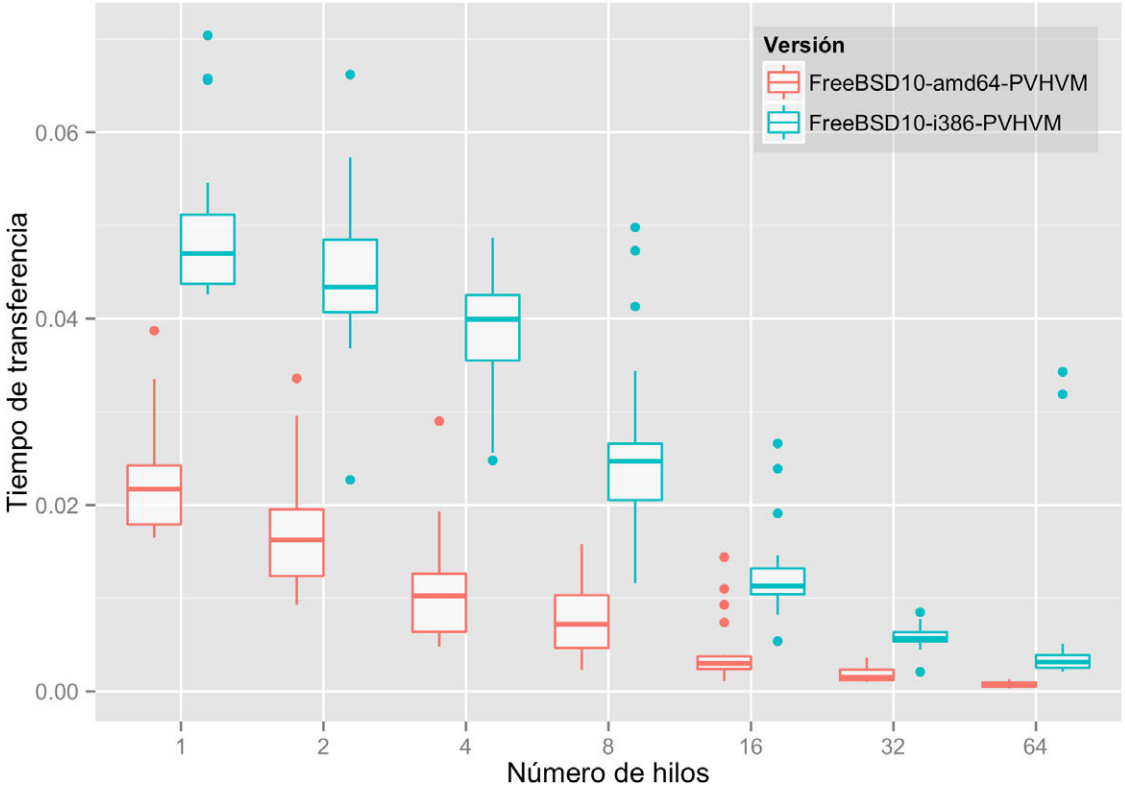

Fig. 7.5. Resultados del test de memoria para PVHVM amd64 vs i386

7.3. Disco

El rendimiento de acceso de disco se ha realizado mediante la herramienta Bonnie++, también disponible para FreeBSD como port o paquete binario.

A diferencia del resto de pruebas, la escritura en disco es sensible a alteraciones permanentes en el sistema de ficheros que podrían disminuir la fiabilidad de los resultados. Para realizar la prueba de la forma más homogénea posible, se optó por crear un dispositivo de bloques dedicado. Para todas las máquinas y cada vez que arranca la prueba, se crea un nuevo disco duro virtual y se le da formato. Se obtiene así un sistema de archivos aislado del resto del sistema operativo y exactamente igual para cada máquina virtual.

Dado que las máquinas del test son instancias virtuales, crear y destruir dispositivos virtuales es relativamente fácil. En Xen Server, el almacenamiento se gestiona mediante una serie de objetos que deben existir antes de poder asignar directamente un nuevo dispositivo a la máquina.

En el primer nivel, están los Storage Repositories. Son entidades de almacenamiento donde se guardan los discos virtuales. Xen soporta distintos tipos de SRs en función del dispositivo físico donde se almacene la información, Pueden ser discos locales, almacenamiento remoto NFS o iSCSI. Los SRs encapsulan toda la lógica necesaria para gestionar cada tipo de almacenamiento.

Las imágenes de discos virtual o VDIs son las unidades fundamentales de almacenamiento en Xen. Su contenido está almacenado de forma persistente dentro de un SR.

Finalmente están los VBD, o virtual block devices. Son los dispositivos virtuales mediante los cuales las máquinas acceden de forma transparente al almacenamiento de bloque. Cada VBD está asociado a un VDI determinado.

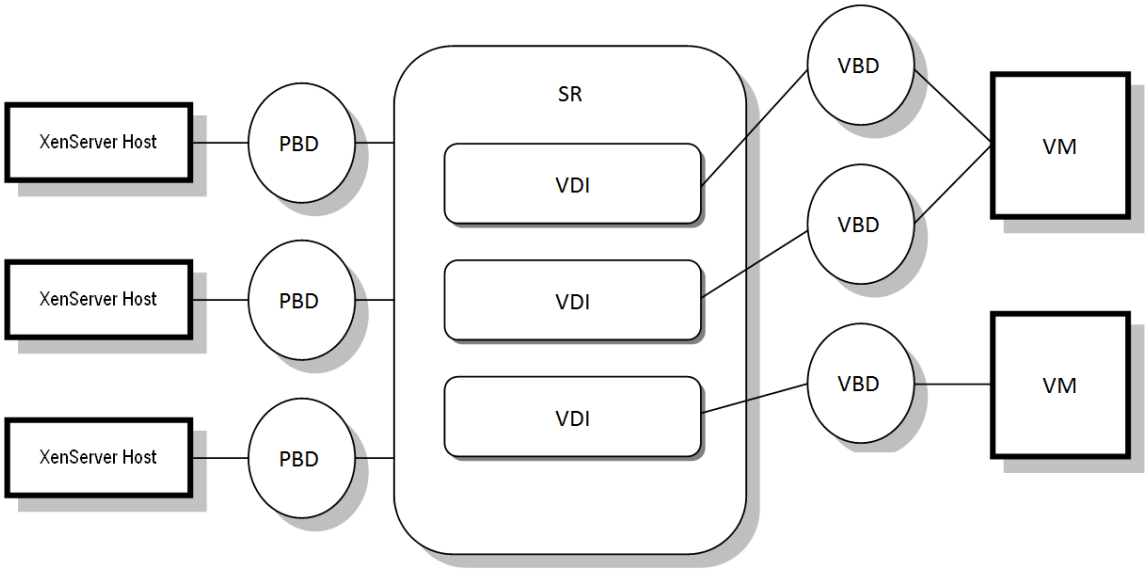

Fig. 7.6. Relaciones entre las entidades lógicas de almacenamiento en Xen

El objetivo para la prueba es que las máquinas dispongan de un dispositivo de bloque extra. Para ello se crea primero una nueva imagen de disco o VDI y se socia a el Storage Repository existente, basado en el dispositivo de disco local del servidor. Para la imagen se ha elegido un tamaño arbitrario de 2GB. Una vez disponible el VDI, éste se asocia como un nuevo VBD a una máquina virtual existente. El VBD aparecerá como un nuevo dispositivo de disco en la máquina virtual.

Los sistemas operativos paravirtualizados pueden hacer esta conexión en caliente. No obstante, dado que sólo algunas máquinas del test están optimizadas para funcionar con Xen, se ha optado por realizar el proceso de creación del nuevo dispositivo con la máquina apagada para todos los casos, independientemente del tipo de máquina virtual.

En cuanto a los resultados, se tomaron muestras para 20 ejecuciones del test empleando un tamaño de archivo de 1GB, sobre el disco virtual dedicado.

A continuación se presentan los resultados en gráficas de caja. Los valores medios de las muestras se han recopilado en dos tablas. Se recogen los indicadores de escritura, re-escritura y lectura. Para cada operación se da la tasa en KB/s y la carga de CPU asociada.

El modo PVHVM se ha incluido en los resultados para i386 y amd64.

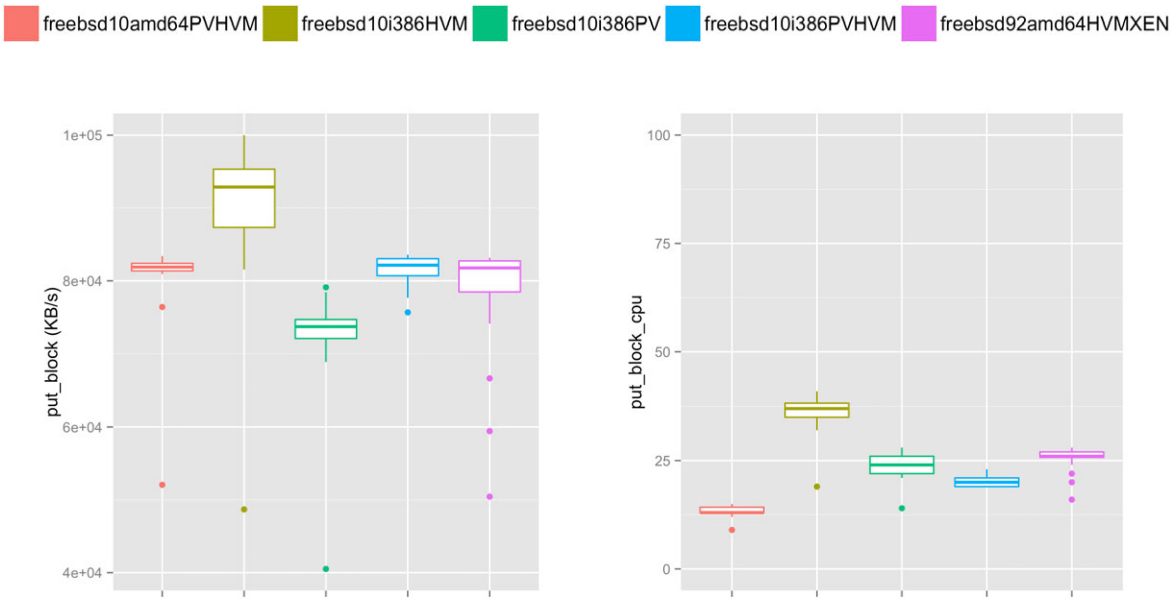

Fig. 7.7. Resultados de la herramienta Bonnie++ para escritura de bloques

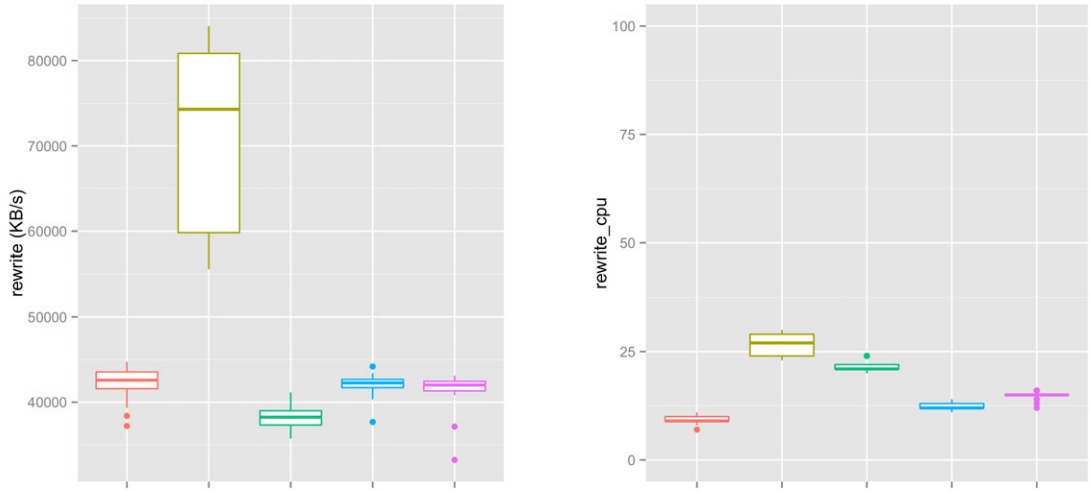

Fig. 7.8. Resultados de Bonnie++ para la reescritura de bloques

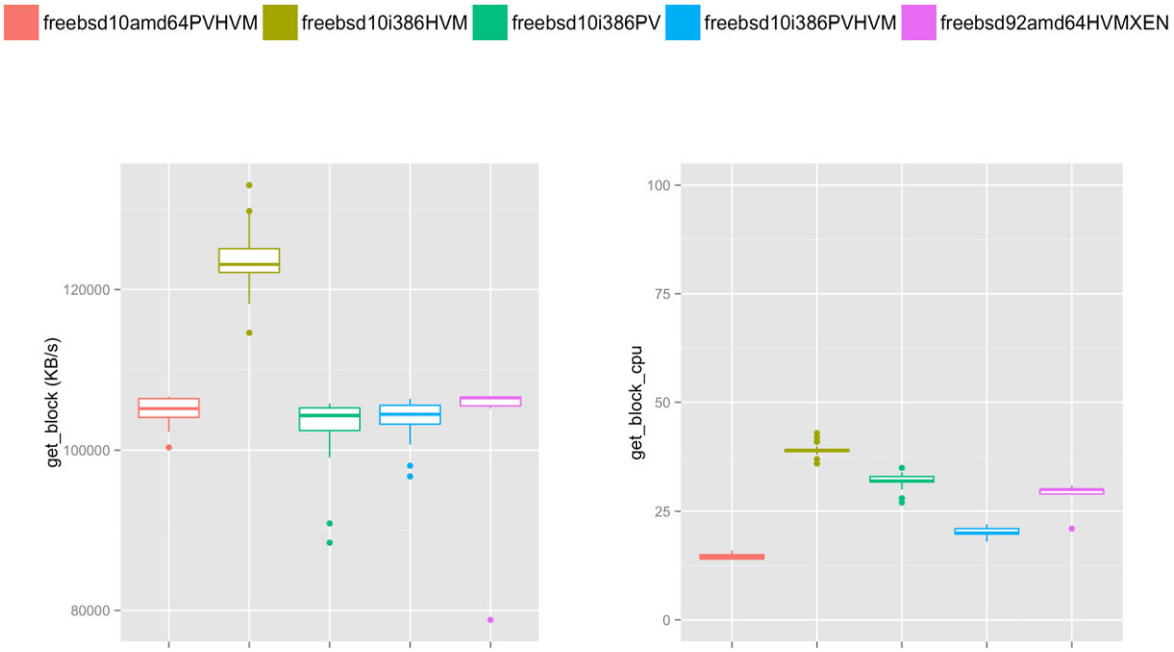

Fig. 7.9. Resultados de la herramienta Bonnie++ para la lectura de bloques

	HVM i386	PV i386	HVMXEN amd64	PVHVM amd64	PVHVM i386
put_block (KB/s)	89664	72206	77916	80313	81437
put_block_cpu (%)	36	24	25	13	20
rewrite (KB/s)	70889	38371	41323	42257	42002
rewrite_cpu (%)	27	21	15	9	12

Tabla 7.4. Resultados de escritura de disco en valor medio.

	HVM i386	PV i386	HVMXEN amd64	PVHVM amd64	PVHVM i386
get_block (KB/s)	123842	102647	104762	104889	103655
get_block_cpu (%)	39	32	29	15	20

Tabla 7.5. Resultados de lectura de disco en valor medio.

La herramienta Bonnie ofrece resultados para operaciones escritura y lectura en disco a nivel de bloque y a nivel de carácter, además de otras informaciones como la latencia o resultados de operaciones aleatorias. Por simplicidad, se han tomado únicamente los valores obtenidos relacionados con la escritura de bloque.

En cuanto a las tasas obtenidas, la máquina no optimizada, con virtualización asistida por hardware ha obtenido los resultados más altos en todos los casos. Por contra muestra mayor fluctuación en la tasa y mayor uso de CPU que las otras implementaciones.

A excepción de la re-escritura de bloques, donde la máquina HVM casi duplica al resto, en general, las tasas obtenidas entre todas las versiones son muy similares.

El uso de CPU se muestra más ligado al tipo de implementación. Es destacable que en modo paravirtualizado no sea el que menos CPU consume, sino los modos híbridos. Los modos PVHVM logran el menor uso de CPU y aunque no superan las tasas del HVM, superan al sistema paravirtualizado.

7.4. Red

Los test realizados con la herramienta iperf tienen como finalidad medir el rendimiento máximo en bits por segundo que el sistema operativo es capaz de obtener de su interfaz de red.

Se realizaron dos pruebas distintas empleando en un caso el protocolo TCP y en el otro transmisión UDP. En todas las pruebas el trafico se origina en la máquina virtual y se dirge a un servidor remoto ubicado en una máquina externa conectada a través de un enlace dedicado ethernet gigabit, capaz de procesar un máximo teórico de 1000mbps full duplex, a nivel de enlace. Para cada escenario tomaron 300 muestras a intervalos de un segundo.

Rendimiento de red para tráfico TCP

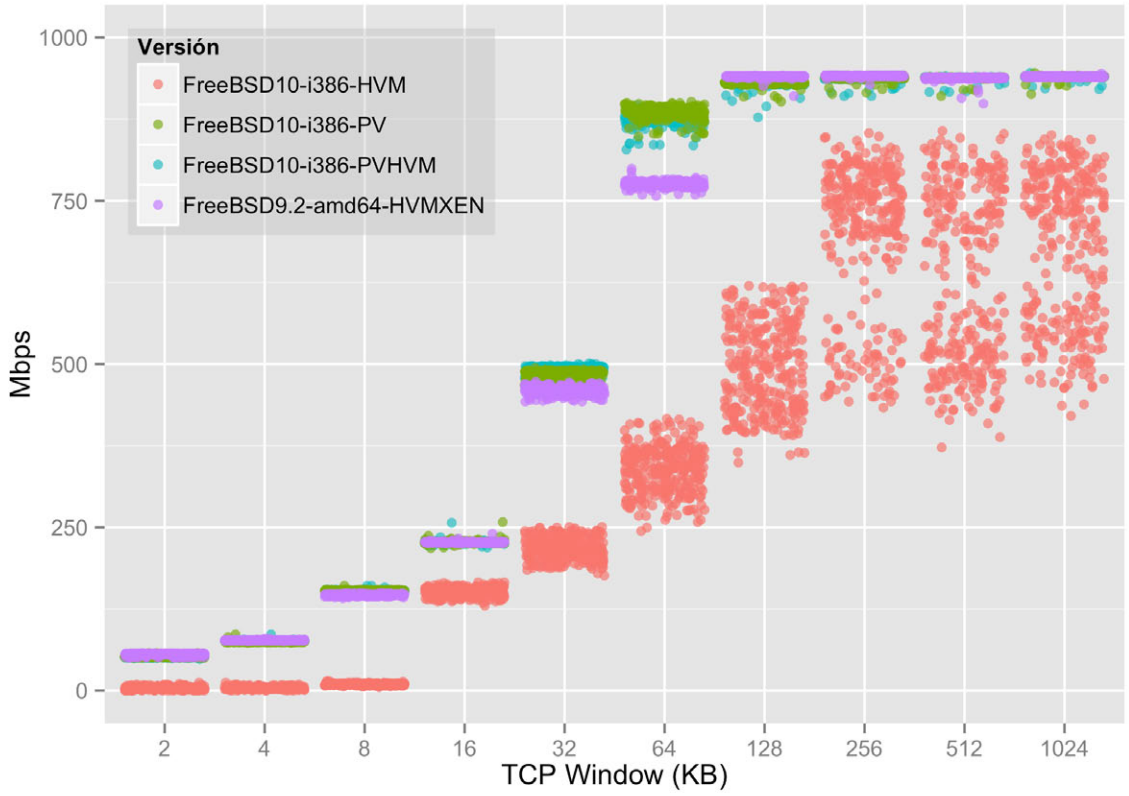

Fig. 7.10. Muestras obtenidas para el test de red TCP

En el caso de la prueba de tráfico TCP, debido a que la tasa de bytes totales transmitidos es altamente dependiente del tamaño de ventana empleado, se realizaron test para distintos valores posibles de ventana TCP. Este tamaño de ventana puede ser ajustado automáticamente por las partes o emplear valores por defecto. Fijando valores concretos por un lado se asegura una prueba consistente y por otro se obtiene una visión más amplia y efectiva del comportamiento TCP.

Observando los resultados puede apreciarse como a excepción del sistema virtualizado por hardware, las muestras son muy consistentes para cada uno de los valores de ventana medidos.

La instancia HVM no sólo ofrece un rendimiento considerablemente inferior al resto sino que ademas presenta una gran varianza entre las muestras obtenidas. Esta máquina virtual a diferencia de las demás carece de cualquier integración con el hipervisor Xen y sus interfaces de red dependen por completo de los drivers emulados que le gestiona el Dom0.

El Dom0 no deja de ser otra instancia Xen como las demás. Obtiene tiempo de ejecución y recursos hardware cuando le corresponden. Cabe pensar por tanto que esta inconsistencia en el rendimiento es debida a que los paquetes de red sufren una penalización mayor en todo el proceso en comparación a aquellos que son gestionados directamente por drivers del DomU FreeBSD.

En la siguiente gráfica, se han reducido las muestras a valores medios. Tras esta simplificación de los datos puede apreciarse como prácticamente todas las implementaciones de FreeBSD con algún tipo de integración con Xen consiguen alcanzar una tasa cercano al máximo teórico para ventanas TCP razonablemente grandes.

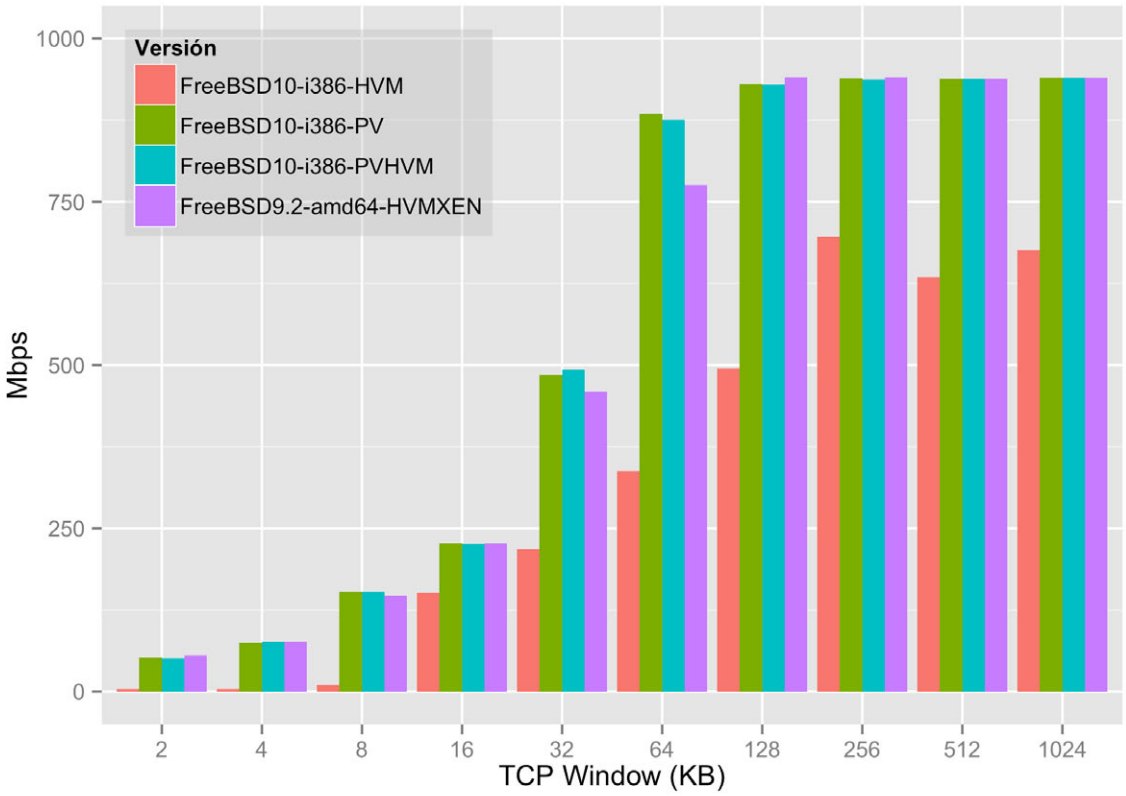

Fig. 7.11. Resultados en valor medio para el test de red TCP

La ventana TCP determina la cantidad de paquetes sin confirmar que pueden ser enviados. Cuanto más rápido es un enlace antes puede alcanzarse esta cantidad. Hasta que la confirmación no es recibida por el emisor, éste deja en enviar paquetes. La capacidad de un enlace puede quedar infrautilizado si no se emplea el tamaño de ventana adecuado. En el caso un interfaz de red ethernet gigabit, se deben emplear al menos valores superiores a 256KB. Inicialmente el protocolo TCP no admitía valores superiores a 64KB pero fue extendido en la RFC1323[4] para poder soportar enlaces más modernos. FreeBSD solía tomar como valor por defecto una ventana de 64KB pero en realidad el kernel incluye una serie de valores mínimo y máximo sobre los que luego se ajusta la transferencia de las sesiones TCP activas.

[4] http://tools.ietf.org/pdf/rfc1323.pdf

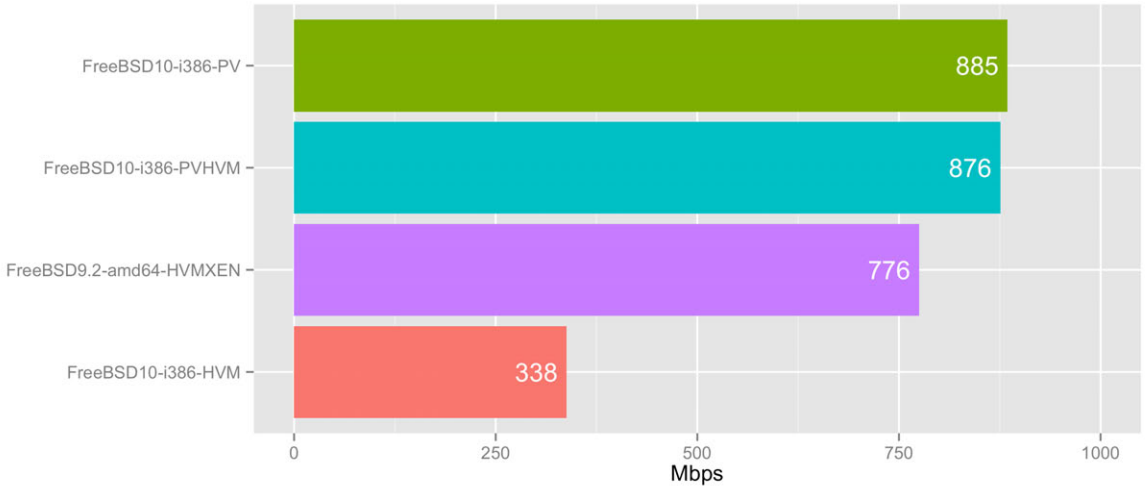

Fig. 7.12. Resultados en valor medio para ventana típica TCP de 64KB

En la gráfica, se han aislado los resultados obtenidos en valor medio tomando la ventana TCP de 64KB como valor típico de referencia. Se observa una diferencia mínima entre el kernel con paravirtualización clásica y el paravirtualización híbrida, que consiguen los rendimientos más altos entre las implementaciones puestas a prueba con unos valores del 88% y 87% respectivamente. No muy por debajo de los más rápido queda el kernel generico con drivers Xen alcanzando el 77% del rendimiento del enlace.
Con una tasa menor al 34%, la máquina FreeBSD con virtualización totalmente asistida por hardware evidencia una clara ineficiencia en la gestión y rendimiento del dispositivo red.

A la vista de los resultados puede concluirse que las implementaciones paravirtualizadas ofrecen los mejores rendimientos siendo notable el hecho de que la implementación PVHVM iguale los resultados de la paravirtualización pura.

La implementación HVMXEN, para valores de ventana TCP adecuados, parece que es capaz de igualar a los modos más avanzados. Sin embargo, como veremos a continuación en los resultados de tráfico UDP, los driver Xen no están a la misma altura que los paravirtualizados. Este alto rendimiento obtenido aquí puede deberse a la idoneidad de la prueba sintética TCP y no ser reflejo del rendimiento con tráfico real que ofrece esta opción FreeBSD sobre Xen.

Rendimiento de red para tráfico UDP

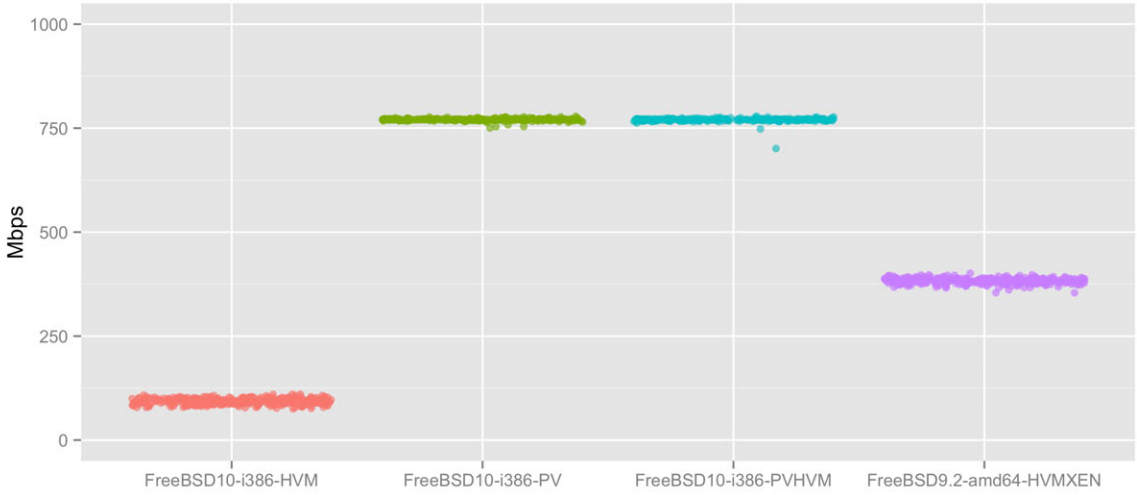

Figura 7.13. Muestras obtenidas para el test de red UDP

Complementario al test de tráfico de red TCP se realizó el mismo experimento con tráfico UDP. Sin control de flujo ni sobrecarga de paquetes de control, el tráfico UDP puede ofrecer una idea más representativa del comportamiento generico del dispositivo y la pila de red del sistema operativo.

Para cada escenario se tomaron otra vez 300 muestras a intervalos de un segundo. A diferencia de TCP no hay mecanismos de control y no se hace por tanto ninguna otra clasificación más allá de la tasa efectiva alcanzada. Esta tasa se fija en envío al límite máximo del enlace: 1000mbps.

Los resultados obtenidos ofrecen unas tasas algo menores a las que se midieron en los test TCP bajo condiciones idóneas. En cuanto al rendimiento, el ranking entre las distintas versiones se mantiene igual con tasas de rendimiento similares a las obtenidas sobre TCP.

El kernel HVM no muestra aquí la fluctuación e inconsistencia que mostraba sobre TCP siendo mucho más consistente el tráfico UDP. Aparentemente, la penalización de esta implementación basada en hardware emulado en el Dom0 afecta más al propio protocolo TCP que al dispositivo de red en sí.

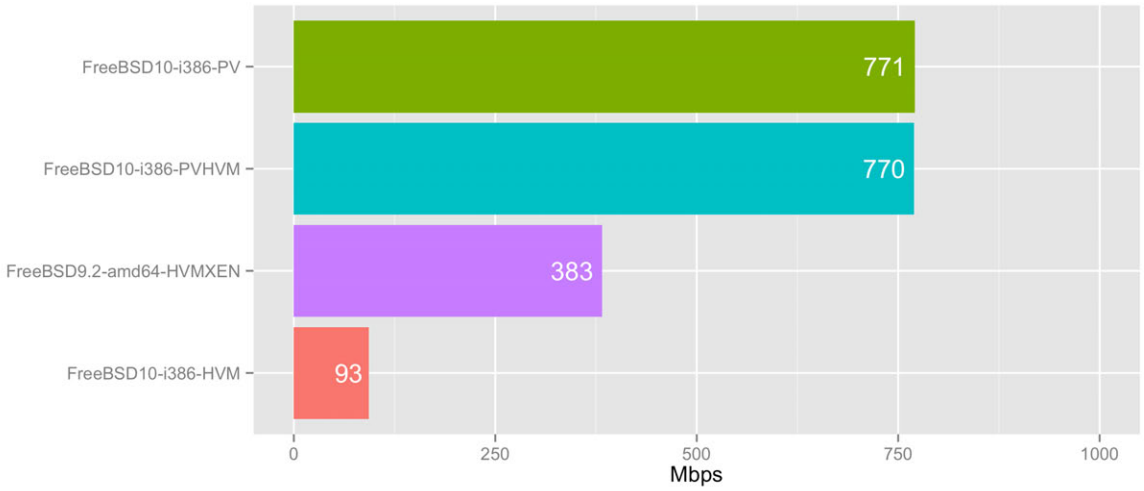

Fig. 7.14. Resultados en valor medio para el test de red UDP

FreeBSD 10 PVHVM amd64 vs i386

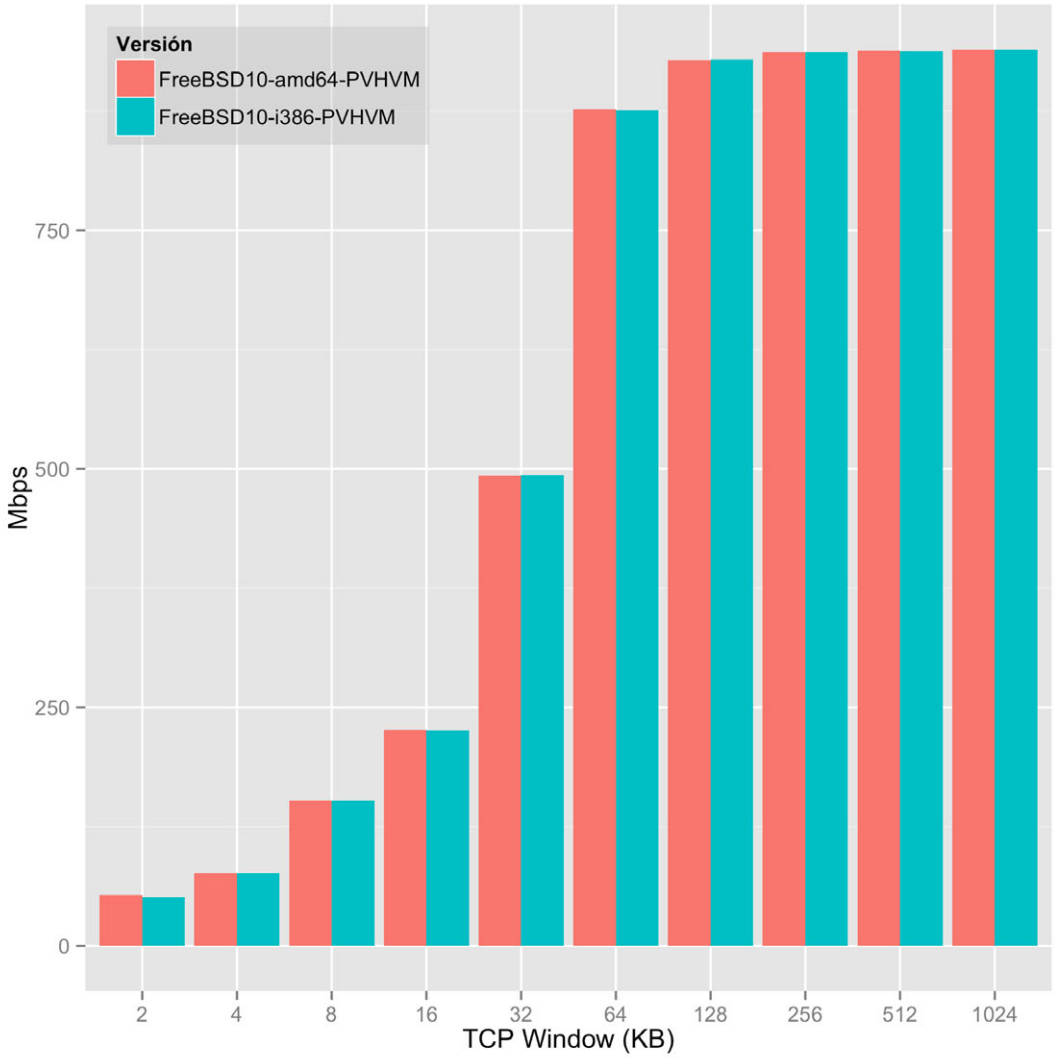

Fig. 7.15. Valores medios para el test de red TCP para FreeBSD10 PVHVM AMD64 vs i386

En el test TCP quedo claro que el rendimiento para implementaciones PV y PVHVM era casi igual. Dado sólo es posible obtener FreeBSD PV para la arquitectura i386 pero para cabe preguntarse si la versión de amd64 PVHVM tiene un rendimiento de red aún mejor.

Como puede observarse en la gráfica entre ambas arquitecturas del kernel de FreeBSD sobre PVHVM no hay diferencias aparentes.

7.5. Sumario

Fig. 7.16. Comparativa de resultados por tipo de virtualización

En la figura 7.16 se han recopilado los valores medios para cada tipo de test. El todo verde indica el mejor valor obtenido, siendo inverso para el caso de los test de CPU y Memoria.

Como puede observarse las implementaciones con resultados más equilibrados son las que incluyen paravirtualización.

En arquitectura i386 la paravirtualización híbrida logra resultados muy similares a los del sistema paravirtualizado, con ligeras ventajas gracias a las funciones que son realizadas por hardware.

En el caso del modo PVHVM de 64bits, los resultados son aún más altos que en el de 32 bits siendo este el modo con más rendimiento más equilibrado de toda la comparativa.

8. Conclusiones

La posibilidad de virtualizar FreeBSD sobre Xen es una combinación de gran potencial que alberga la incertidumbre propia de los entornos virtualizados en cuanto al rendimiento efectivo que puede verse afectado.

En este trabajo se han estudiado los distintos mecanismos de virtualización y se han puesto en práctica en las implementaciones de FreeBSD compatibles con Xen identificadas.

Los resultados obtenidos han permitido tener una visión completa de las posibilidades que ofrece cada uno de las implementaciones disponibles destacando los puntos fuertes de conjunto de opciones.

A la vez, este estudio ha servido para constatar de forma práctica y aplicada a FreeBSD la propia evolución de la tecnología de virtualización, exponiendo las posibilidades de la paravirtualización y el hipervisor Xen.

9. Trabajo futuro

En este proyecto se ofrece una visión del estado actual de la virtualización de FreeBSD sobre Xen centrándose en un conjunto limitado de pruebas y de configuraciones.

Por un lado, la tecnología continúan en constante evolución. El ecosistema Xen y FreeBSD seguirán introduciendo mejoras y ofreciendo nuevas versiones cuyo rendimiento sera interesante conocer. Próximamente Xen 4 incluirá el soporte de paravirtualización asistida por hardware, aún experimental en FreeBSD. Este modo ofrecerá previsiblemente un rendimiento aún mejor al de la virtualización híbrida actual.

Por otro lado, en lo relativo a las pruebas realizadas aquí, el ámbito y el método también puede ampliarse.

Una de las pruebas que no se ha cubierto en este trabajo es la medida del rendimiento de FreeBSD sin virtualizar, realizando el mismo conjunto de pruebas y medidas. El rendimiento obtenido puede tomarse como referencia para después normalizar los resultados de las implementaciones virtualizadas. Esto en principio requiere mayor infraestructura, con dos sistemas exactamente iguales. También sería posible compartir el sistema de pruebas empleando un gestor de arranque. El programa de pruebas podría adaptarse para, en función del tipo de máquina —virtual o real, interactuar con el gestor de arranque de forma que se arranquen imágenes sin virtualizar, directamente en el hardware, o máquinas virtuales dando paso primero al arranque del hipervisor.

Otro punto interesante sería el de la automatización de la instalación del sistema operativo para aumentar la fiabilidad de las pruebas. En este trabajo la instalación de los sistemas se ha se ha realizado una única vez, de forma manual, y sobre estas instalaciones se han pasado todas las pruebas. El programa de pruebas podría extenderse para instalar cada vez el sistema operativo hasta el punto de crear él mismo una nueva máquina virtual en Xen, realizando después todo el

proceso de instalación y configuración de FreeBSD. Algo similar podría lograrse conservando un clon de la máquina virtual instalada y cada que vez que se lance una prueba, crear una máquina virtual a partir de una copia de la imagen preinstalada.

En todos los casos de prueba se ha ejecutado una sola máquina virtual puesto que funcionando aislada era la forma adecuada de conocer su rendimiento. Pudiendo generar más instancias de forma automática sería interesante conocer la degradación que se produce dentro de la máquina virtual cuando el número de maquinas virtuales con las que comparte recursos aumenta. Aunque este aspecto es más un indicador del hipervisor que de la propia maquina virtual, modelando pruebas de escalabilidad con sistemas virtualizados exactamente iguales puede minimizarse la incertidumbre que ya existe de por si en un sistema tan complejo y ayudar en la extracción de conclusiones. El sistema de pruebas podría extenderse para ejecutar ciclos de pruebas que vayan añadiendo más máquinas virtuales, a la vez que se registra su rendimiento.

También ha quedado fuera del ámbito probar con más de una CPU virtual. Xen soporta la asignación estática o dinámica de múltiples CPUs virtuales a cada uno de los dominios virtuales de usuario. Aunque FreeBSD aún tiene problemas en algunas de sus implementaciones, como es el caso de la paravirtualizada, la versión genérica virtualizada por hardware puede funcionar con más de una CPU. El sistema de pruebas podría modificarse para realizar baterías de pruebas que jueguen con la asignación de vCPUs en las instancias Xen para obtener resultados que ayuden a conocer cual es la distribución óptima de CPUs virtuales.

10. Referencias

Artículos

Goldberg, R. P. (1974). Survey of virtual machine research. Computer, 7(6), 34–45

Popek, G. J., & Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. Communications of the ACM, 17(7).

Barham et al. (2003). Xen and The Art of Virtualization. SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles

Hand et al. (2007). Hardware virtualization with Xen. Login: The USENIX Magazine, 21-27.

VMWare. (2007). Understanding Full Virtualization, Paravirtualization, and Hardware Assist, 1–17.

Huber, N., Quast, Von, M., Brosig, F., & Kounev, S. (2010). Analysis of the performance-influencing factors of virtualization platforms. Presented at the OTM'10: Proceedings of the 2010 international conference on On the move to meaningful internet systems: Part II, Springer-Verlag.

Huber, N., Quast, von, M., & Hauck, M. (2011). Evaluating and modeling virtualization performance overhead for cloud environments.

Nakajima, J., Lin, Q., Yang, S., Zhu, M., & Gao, S. (2011). Optimizing virtual machines using hybrid virtualization.

Ediciones

Chisnall, D. (2007). The Definitive Guide to the Xen Hypervisor (1ª ed.). Prentice Hall.

Lucas, M. (2007). Absolute FreeBSD (2ª ed.). San Francisco: No Starch Press.

Takemura, C., & Crawford, L. S. (2009). The Book of Xen (1ª ed.). San Francisco: No Starch Press.

Recursos Web

Archivo de la lista de correo FreeBSD-Xen, 2008-2014
http://lists.freebsd.org/pipermail/freebsd-xen/

FreeBSD and Linux Kernel Cross-Reference
http://fxr.watson.org

The Xen Project Blog
http://blog.xenproject.org

The Xen Project Wiki
http://wiki.xen.org/wiki/Xen_Project_Software_Overview

Citrix XenServer ® 6.2.0 Administrator's Guide
http://docs.vmd.citrix.com/XenServer/6.2.0/1.0/en_gb/reference.html

Herramienta Sysbench
https://launchpad.net/sysbench

Herramienta Bonnie++
http://www.coker.com.au/bonnie++

Herramienta iperf
https://iperf.fr

The R Project for Statistical Computing
http://www.r-project.org